UMA INTERFACE GRÁFICA DE USUÁRIO PARA PROJETO E IMPLEMENTAÇÃO DE FILTROS DIGITAIS COM PROCESSADOR DEDICADO

Arnaldo Cordeiro Machado

Dissertação submetida ao corpo docente do Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal do Espírito Santo como parte dos requisitos necessários para a obtenção do título de Mestre em Engenharia Elétrica.

Aprovada em 19/06/96 por:

_____

Prof. D. Sc. Mário Sarcinelli Filho - UFES
Orientador

_____

Prof. Dr.rer.nat Hans-Jörg Andreas Schneebeli - UFES
Co-orientador

_____

Prof. D. Sc. Luis Claudius Coradine - UFES

_____

Prof. D. Sc. Mauro Cavalcante Pequeno - UFC

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
VITÓRIA, ES - BRASIL
Junho de 1996

MACHADO, ARNALDO CORDEIRO, 1970
Uma Interface Gráfica de Usuário para Projeto e Implementação de Filtros Digitais com Processador Dedicado [Vitória] 1996.
vi, 125 pp., 29.7 cm (PPGEE/UFES, M.Sc., Engenharia Elétrica, 1996)
Dissertação, Universidade Federal do Espírito Santo, PPGEE.
I. Processamento Digital de Sinais
I. PPGEE/UFES II. Título (Série)

AGRADECIMENTOS

Aos amigos que de alguma maneira me ajudaram e incentivaram a concluir esse trabalho, dentre professores, colegas e familiares. Em particular aos meus pais, e, a querida Kátia Frassi de Souza.

RESUMO DA DISSERTAÇÃO APRESENTADA AO PPGEE/UFES COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA (M. Sc.)

UMA INTERFACE GRÁFICA DE USUÁRIO PARA PROJETO E IMPLEMENTAÇÃO DE FILTROS DIGITAIS COM PROCESSADOR DEDICADO

Arnaldo Cordeiro Machado
junho de 1996

ORIENTADORES: Mário Sarcinelli Filho
Hansjorg Andreas Schneebeli

PROGRAMA: Engenharia Elétrica

Este trabalho descreve uma plataforma de software desenvolvida como um ambiente para o projeto e a implementação de filtros digitais usando DSP. O sistema foi desenvolvido e é executado em ambiente WINDOWS, para microcomputadores PC dotados de uma placa de desenvolvimento baseada em DSP. A placa de DSP utilizada é baseada no "chip" TMS320C25, de ponto fixo, 16 bits, da Texas Instruments, acoplada ao barramento ISA do microcomputador. São três as características básicas do sistema desenvolvido: o projeto dos filtros a serem implementados, restrito ao processador do computador PC, a execução em tempo real dos filtros, restrita ao DSP, e a troca de informações entre os processadores, ou entre o computador PC e o usuário, para a programação dos coeficientes dos filtros a serem implementados e para a monitoração dos sinais de entrada/saída. Adicionalmente, o usuário pode editar/visualizar arquivos de coeficientes das redes (ou das funções de transferência) a serem implementadas, assim como editar/compilar/aperfeiçoar arquivos contendo programas "assembly" para o DSP.

ABSTRACT OF THESIS PRESENTED TO PPGEE/UFES AS PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER IN ELECTRICAL ENGINEERING (M. Sc.)

GRAPHICAL USER INTERFACE FOR THE DESIGN AND DSP-BASED IMPLEMENTATION OF DIGITAL FILTERS

Arnaldo Cordeiro Machado
june 1996

THESIS SUPERVISORS: Mário Sarcinelli Filho
Hansjorg Andreas Schneebeli

DEPARTMENT: Engenharia Elétrica

This work describes a software platform developed in order to support the design and DSP-based implementation of digital filters. It is a WINDOWS program designed to run on a personal computer containing a DSP board. A card based on the 16-bit fixed point Texas Instruments TMS320C25 chip is used, which is attached to the ISA PC bus. The developed software is specially designed to accomplish three main features: the filter design task, restricted to the PC microprocessor, the real-time synchronous filter algorithm, running exclusively on the DSP, and the necessary information exchanging between both processors, and between the PC computer and the user, in order either to load filter parameters or to check for the filter input/output signals. In addition, the user may edit/view text files containing the filter transfer function or network coefficients, or edit/compile/debug files that contain assembly code for the DSP.

Figuras

Tabelas

CAPÍTULO 1

Introdução

1.1. Processamento digital de sinais usando processadores dedicados

O uso de microprocessadores dedicados, os DSP's (Digital Signal Processors), é uma tecnologia já bastante sedimentada para a implementação de filtros digitais. Por possuírem arquitetura do tipo Harvard e serem dotados de hardware específico para a operação de multiplicação /Tex93/, os DSP's têm como principal característica uma arquitetura particularmente adaptada para a realização das operações de multiplicação e acumulação de produtos, típicas de filtros digitais, permitindo realizar tais operações em um tempo muito pequeno /Cha90/. Como conseqüência, a implementação de uma determinada familia de filtros digitais usando DSP passa a ser bastante simplificada, reduzindo-se à escrita de um programa fonte, na linguagem "assembly" particular do DSP ou em uma linguagem de alto nível especialmente projetada para tal microprocessador (vários DSP's admitem programas em linguagem C /Tex93/).

Dentre os vários DSP's existentes no mercado, aqueles que usam aritmética de ponto fixo e representação numérica em complemento a dois são os mais comumente usados para a implementação de filtros digitais /Mul76a, Laa87, Bom94/. Isto é devido ao fato de que normalmente poucos bits são usados na implementação dos referidos filtros, pois, na maioria das vezes, a faixa dinâmica necessária à representação numérica é pequena (freqüentemente, os valores resultantes do processamento possuem módulo inferior à unidade) /Sar90, Sim94/. Entretanto, alguns resultados com implementação em ponto flutuante têm sido recentemente reportados /Bau93, Laa94a, Laa94b, Kim94, Bau95/, muito embora também já tenham sido reportados outros resultados anteriores à era dos DSP's /Kan73/.

No presente trabalho será usado o DSP TMS320C25, da Texas Instruments, o qual opera em ponto fixo e processa dados de 16 bits, possuindo dois registradores (o acumulador e o registrador P para o armazenamento do resultado dos produtos) de precisão dupla /Tex93/. Devido ao fato de que na grande maioria das vezes os coeficientes multiplicadores adotados possuem módulo inferior à unidade /Sar90, Sim94/ (como é o caso da implementação das estruturas de filtros digitais no espaço de estados) isto leva a um erro de representação muito pequeno, tanto dos coeficientes como dos sinais /Cha90/, já que 15 bits podem ser usados para representar a mantissa.

Para amostrar um sinal no tempo contínuo, gerando uma seqüência numérica, processar essas amostras e transformá-las novamente em um sinal no tempo contínuo, utiliza-se técnicas de processamento digital de sinais. É importante também ser destacado que em muitas aplicações os dados já se encontram disponíveis na sua forma digital. A Figura 1.1, a seguir, ilustra os principais componentes envolvidos no processamento digital de sinais /Opp89, Ant93, Rab75, Sta84/.

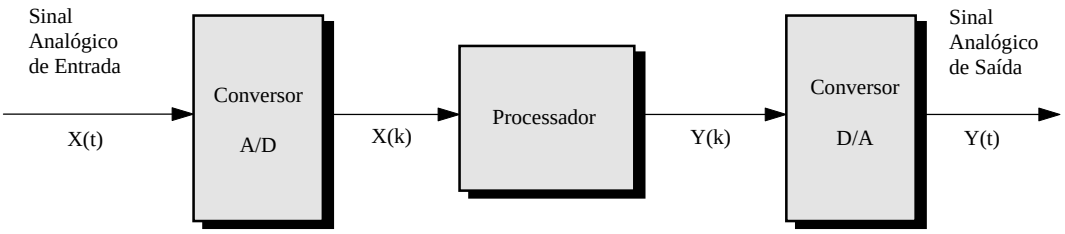


Figura 1.1. Sistema típico para processamento digital de sinais.

Ao longo desses últimos anos as técnicas e as aplicações do processamento digital de sinais vêm se expandindo. Em algumas áreas houve uma verdadeira revolução, como é o caso da área de telecomunicações, onde o emprego dessas técnicas está trazendo melhorias substanciais em termos de economia e flexibilidade nas implementações de sistemas de comutação e transmissão /Sta84/. Em outras áreas, como as de audio digital, engenharia biomédica, sistemas de sonar e radar, instrumentação, processamento de imagens, processamento de sinais sísmicos, etc., as técnicas de processamento digital de sinais também estão trazendo grandes benefícios.

O avanço da tecnologia da microeletrônica, mais especificamente a de VLSI ("Very Large Scale Integration"), passou a ser uma das principais razões para o crescimento do emprego de técnicas de processamento digital de sinais, trazendo, consequentemente, uma sensível redução de custo e tamanho de componentes, sem contar o aumento em termos de velocidade de processamento /Ess86/.

Os computadores digitais de uso genérico favoreceram enormemente o crescimento nessas áreas, possibilitando o desenvolvimento de algoritmos complexos para processamento de sinais. Porém, em aplicações que requeriam processamento em tempo real, tais computadores ainda levavam um tempo excessivamente grande para processar os dados. Os microprocessadores digitais de uso específico, com sua grande capacidade de processamento, solucionaram boa parte dos problemas de processamento em tempo real.

Porém, inicialmente os algoritmos eram implementados exclusivamente com microprocessadores baseados em um hardware especial, que era projetado separadamente para cada aplicação. A partir dos primeiros processadores de sinais, começaram a ser, então, desenvolvidos microprocessadores programáveis que foram otimizados especialmente para a tarefa de processamento de sinais. Isto facilitou, em muito, a tarefa de implementação: os algoritmos puderam ser implementados pela simples alteração do programa do processador.

O surgimento da primeira geração de processadores digitais de sinais foi o passo definitivo para equacionar o problema do processamento em tempo real /Per88/. Esses processadores passaram então a ser largamente utilizados devido às vantagens que ofereciam em relação aos circuitos analógicos; a implementação digital tem a vantagem de evitar alguns problemas inerentes à implementação analógica, tais como:

- confialibilidade;
- reprodutividade;
- flexibilidade.

Desta forma, muitas operações que antes eram consideradas muito difíceis ou completamente impossíveis de serem implementadas analogicamente são agora resolvidas no domínio digital, como por exemplo filtros de banda muito estreita (são usadas estruturas realizadas no espaço de estados, implementadas nas formas cascata ou paralela de blocos de segunda ordem /Sim94/). Os sistemas programáveis proporcionam grande versatilidade de projeto, com uma sensível redução no tempo de desenvolvimento. Além disso, os sistemas digitais são mais fáceis de serem reproduzidos, devido às suas características de sensibilidade, confiabilidade e facilidade de ajustes.

Devido a estas evoluções, as técnicas de processamento digital de sinais estão cada vez mais presentes nos mais variados campos de aplicações.

1.2. Motivação e objetivos

Tendo como elemento básico o processador de sinais TMS320C25, o presente trabalho apresenta um ambiente totalmente integrado para o desenvolvimento de sistemas de processamento digital de sinais. Como características básicas desta plataforma computacional pode-se destacar:

- sua facilidade de uso (baseado em uma interface gráfica);
- sua capacidade de executar todas as operações fundamentais do processamento de sinais em tempo real;
- seu baixo custo de implementação;
- sua integração completa entre os módulos de desenvolvimento e projeto, proporcionando ao projetista de filtros digitais a facilidade de concepção e imediata implementação do filtro projetado.

Embora o desenvolvimento de vários softwares de apoio ao projeto de filtros digitais já tenha sido reportado /Per88/, o objetivo de dotar o projetista de filtros digitais de um sistema totalmente integrado, de múltiplas tarefas, que lhe permitisse realizar todo o trabalho de projetar e implementar os filtros desejados, a nível da tecnologia de processadores dedicados programáveis constituiu a principal motivação que levou ao desenvolvimento da plataforma aqui descrita.

A necessidade de criar uma interface mínima que permitisse carregar na memória acessível ao DSP os coeficientes de um filtro digital particular, permitindo assim que o programa “assembly” utilizado fosse mais genérico, correspondendo a uma familia inteira de filtros (cada filtro em particular caracterizado por seus coeficientes e sua freqüência de amostragem /Sar94/), constituiu o ponto de partida para a implementação da plataforma.

Torna-se necessário, também, salientar que a plataforma desenvolvida está atualmente restrita a algumas estruturas de filtros digitais realizados no espaço de estados /Sim94/. Porém ela é bastante modular, permitindo a inclusão de novas classes de estruturas, pela sua adequada inserção no conjunto, na etapa de síntese do filtro, e através do adequado programa fonte para o DSP.

Com o objetivo de garantir uma maior generalidade ao sistema desenvolvido, uma placa comercialmente disponível, baseada em DSP, é usada. Esta placa, fabricada pela Dalanco Spry, é dotada de conversores A/D e D/A, e de um temporizador externo ao DSP, usado para sincronizar os conversores /Dal90/. Podem ser utilizadas, entretanto, outras placas, até mesmo baseadas em outros DSP's, com a necessária atualização dos parâmetros básicos, tais como o endereço de acesso à placa, no espaço de endereçamento do computador PC, programação do temporizador e dos conversores, dentre outros. A Figura 1.2 ilustra, a nível de diagrama de blocos, uma idéia básica da estrutura da placa usada e da sua conexão ao computador PC.

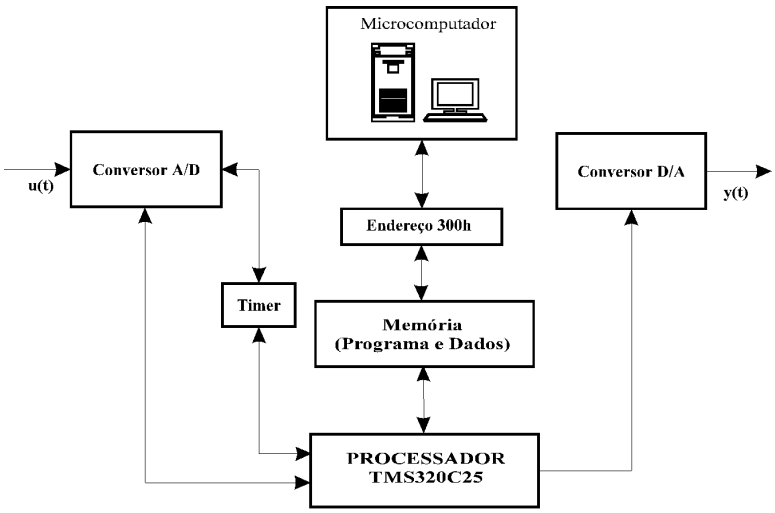


Figura 1.2. Diagrama de blocos da placa de DSP e sua conexão ao computador PC.

Como pode ser observado através da Figura 1.2, estão disponíveis dois processadores: o DSP e a CPU do PC hospedeiro. O DSP, responsável pela execução de um programa "assembly", permite a implementação de algoritmos que caracterizam famílias inteiras de filtros digitais, possuindo os mais variados espectros. O PC por sua vez, fica responsável pela interface com o usuário.

1.3. A plataforma integrada para processamento digital de sinais

O desenvolvimento de filtros digitais requer que certas etapas sejam cumpridas para se ter sucesso na sua implementação. O fluxograma ilustrado pela Figura 1.3 mostra bem essas etapas.

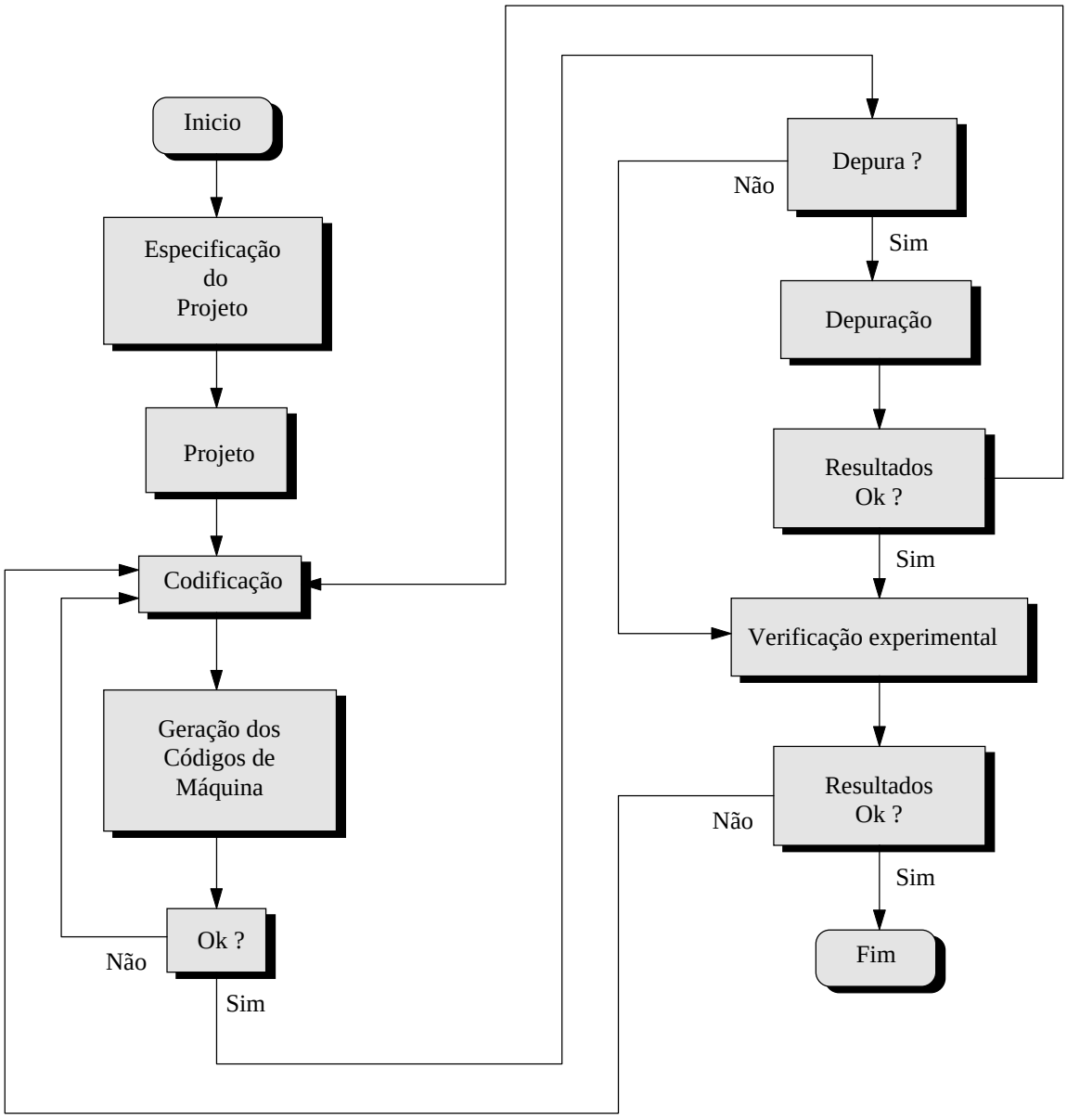


Figura 1.3. Fluxograma das etapas de desenvolvimento.

Com as especificações do filtro, projeta-se então um sistema que as atenda /Tex86/. A partir do projeto completo, passa-se então à fase de codificação dos algoritmos, para que, em seguida, possam ser gerados os códigos de máquina do processador que será utilizado no projeto. A partir desse momento, pode-se fazer uma depuração dos programas desenvolvidos ou ir diretamente para a verificação prática. O ambiente proposto neste trabalho abrange desde a fase de especificação até a fase de verificação experimental, em DSP, da performance do filtro projetado.

A **Plataforma Integrada para Processamento Digital de Sinais - PIPDS** - aqui descrita, tem por principal objetivo agrupar em um único ambiente, totalmente integrado, todo o hardware e software envolvidos no projeto e avaliação experimental de diversas familias de estruturas de filtros digitais, quando implementada usando

tecnologia de processadores digitais de uso dedicado /Cha90/. Ela foi desenvolvida para fazer parte do sistema computacional instalado no **LABSIM - Laboratório de Simulação** do Programa de Pós-Graduação em Engenharia Elétrica da UFES, constituindo-se em um arquitetura gráfica dedicada ao projeto e implementação baseada no processador TMS320C25, da Texas Instruments.

A seguir é feita uma breve apresentação desta arquitetura e dos conceitos básicos envolvidos em sua concepção e execução.

1.3.1. Descrição do ambiente gráfico

Uma das propriedades bastante exploradas em engenharia de software é a modularidade, por proporcionar ao programador a possibilidade de um tratamento independente para cada módulo de um sistema, além de permitir uma visão ampla da organização do software a partir de seu núcleo central /Pre82, Fai85/.

A propriedade de modularidade, no caso da plataforma desenvolvida, é implementada através de um programa principal (núcleo central) que gerencia diversos módulos, os quais, por sua vez, possuem sub-módulos.

Em qualquer projeto de software interativo - que precisa de uma sofisticada interface homem/máquina - o programador não apenas necessita garantir o cumprimento das tarefas para as quais o software foi concebido, mas também a sua capacidade de se comunicar com o usuário /Pre82, Fai85/. É claro que, se os usuários não conseguirem manipular efetivamente o sistema, sua capacidade computacional não será plenamente utilizada, e ele se tornará ineficiente.

Quando da concepção do ambiente integrado de que trata este trabalho, esses dois princípios foram levados em consideração: em outras palavras, a plataforma aqui descrita é modular e apresenta ampla facilidade de interação com o usuário.

1.4. Organização do trabalho

Nos capítulos que se seguem, os detalhes de implementação e funcionamento da plataforma implementada serão apresentados.

No Capítulo 2 são apresentados os conceitos básicos envolvidos na filtragem digital, além de algumas considerações sobre o comprimento finito de palavra, problema inerente à implementação em processadores dedicados, e, em seguida, são revistas as principais estruturas de filtros implementadas na plataforma.

No Capítulo 3 são apresentados detalhes das características e da arquitetura do processador TMS320C25, além de ser discutida a implementação de filtros digitais em processadores digitais de sinais.

No Capítulo 4 são mostradas as técnicas básicas e os principais problemas encontrados na programação baseada em uma interface gráfica, e, por fim, são apresentados os detalhes de implementação da Plataforma Integrada para Processamento Digital de Sinais.

No Capítulo 5 são apresentados os resultados obtidos com a implementação de um exemplo de filtro digital utilizando a plataforma proposta.

Finalmente, no Capítulo 6, são apresentadas as conclusões finais e feitas algumas sugestões para trabalhos futuros.

CAPÍTULO 2

Terminologia e Algoritmos Envolvidos na Filtragem Digital

2.1. Filtros digitais

No domínio digital, um sinal no tempo discreto é usualmente obtido através da amostragem, em intervalos regulares, do sinal original no tempo contínuo. Um sistema digital de ordem N, linear e invariante ao deslocamento, processando sinais no tempo discreto representados numericamente, pode ser definido através da equação diferença

$$y(n) = \sum_{k=0}^{N} b_k . x(n-k) - \sum_{k=1}^{N} a_k . y(n-k) \qquad (2.1)$$

A amostra da saída, y(n) é, portanto, obtida como a soma ponderada de uma entrada, x(n), e dos N valores passados da entrada e da saída, x(n - k) e y(n - k), sendo k = 1, ..., N. Quando os coeficientes a_k são não nulos, o filtro é dito ser do tipo recursivo ou IIR (Infinite Impulse Response), ou seja, um filtro de resposta ao impulso unitário de tempo de duração infinito /Ant93/. Se os coeficientes a_k forem nulos, o filtro é dito ser do tipo não-recursivo ou FIR (Finite Impulse Response). Somente filtros digitais recursivos serão aqui tratados, sem perda de generalidade do trabalho.

O sistema digital também pode ser descrito, agora sob o ponto de vista computacional, como uma rede constituída de somadores, multiplicadores e elementos de atraso. Uma possível rede para realizar a equação (2.1) é mostrada na Figura 2.1.

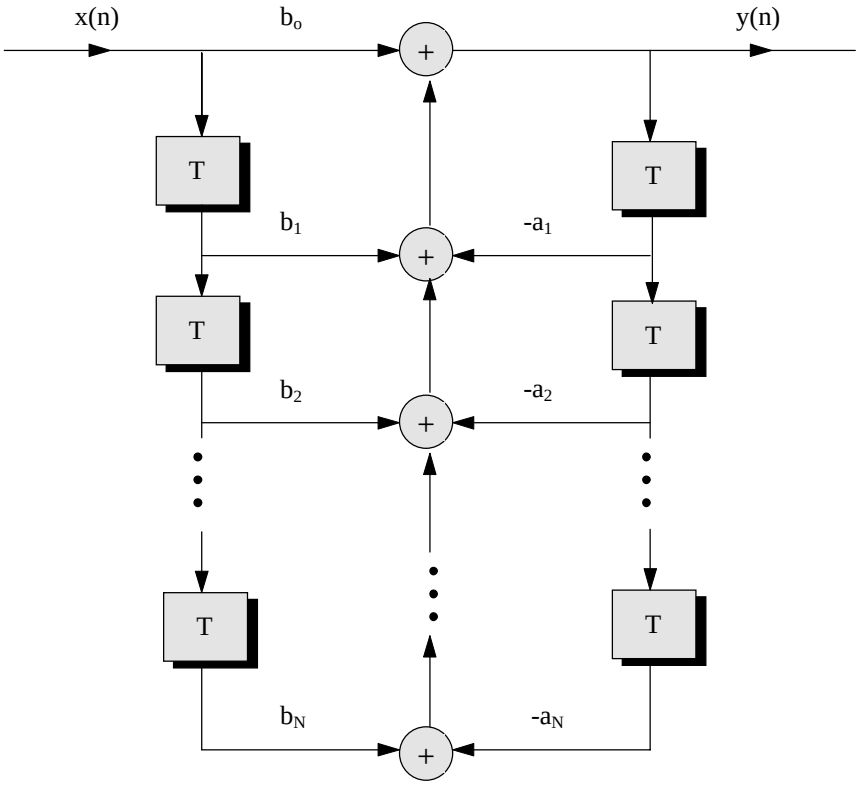


Figura 2.1. Implementação na forma direta da equação diferença (2.1).

Particularmente, o bloco assinalado com T equivale a reter, durante um período de amostragem, o valor apresentado em sua entrada. Isso é feito usando memória, e um sinal de "clock" para sincronização.

Usualmente, as características do sistema no domínio da freqüência são as mais interessantes. Uma poderosa ferramenta, usada para análise no domínio da freqüência, é a Transformada Z, que pode ser vista como uma generalização da transformada discreta de Fourier, ou uma versão discretizada da transformada de Laplace. Assim, a função de transferência correspondente à equação (2.1), no domínio da variável complexa z, pode ser escrita como

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{N} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}} = \frac{b_0 + b_1 z^{-1} + ... + b_N z^{-N}}{1 + a_1 z^{-1} + ... + a_N z^{-N}} \qquad (2.2)$$

após ter sido convenientemente escolhida a região de convergência da transformada z /Ant93/. Observe-se, agora, que o atraso de uma amostra é representado por z^{-1}. A expressão H(z), assim obtida, é usualmente conhecida como a função de transferência do sistema digital.

2.2. O projeto do filtro digital

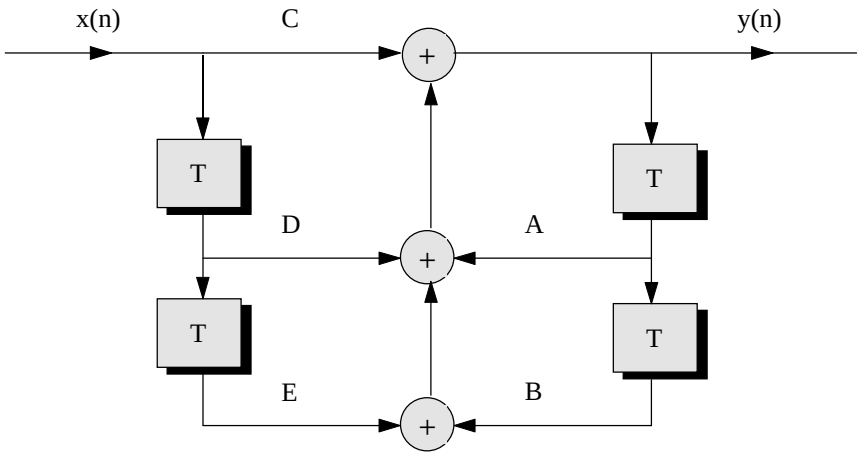
O projeto de um filtro digital consiste, basicamente, em duas etapas. A primeira, dita aproximação, consiste em encontrar a função de transferência H(z), dadas as especificações, com a mínima ordem para o filtro. Existem métodos diferentes para aproximar as especificações. Um dos métodos mais usados para aproximar filtros digitais recursivos é a transformação de protótipos analógicos /Ant93/.

Quando a função de transferência do filtro é obtida (ou seja, um filtro digital IIR linear, invariante ao deslocamento, causal e estável é descrito), necessitamos definir o algoritmo para o seu cálculo, ou seja, a estrutura de rede para realizá-la. Em termos de filtros lineares ideais, todas as diferentes estruturas de redes possuem o mesmo desempenho. Mas, quando o filtro é implementado com um hardware digital qualquer, como ocorre na prática, existem requisitos computacionais que devem ser analisados, tais como a complexidade computacional, considerando o hardware utilizado, e os erros introduzidos devido a cálculos com precisão finita. Neste caso, as estruturas podem apresentar significativas diferenças quanto ao seu desempenho. Portanto, o projetista passa a ter um novo problema, que consiste na seleção de uma estrutura adequada para computar o referido filtro.

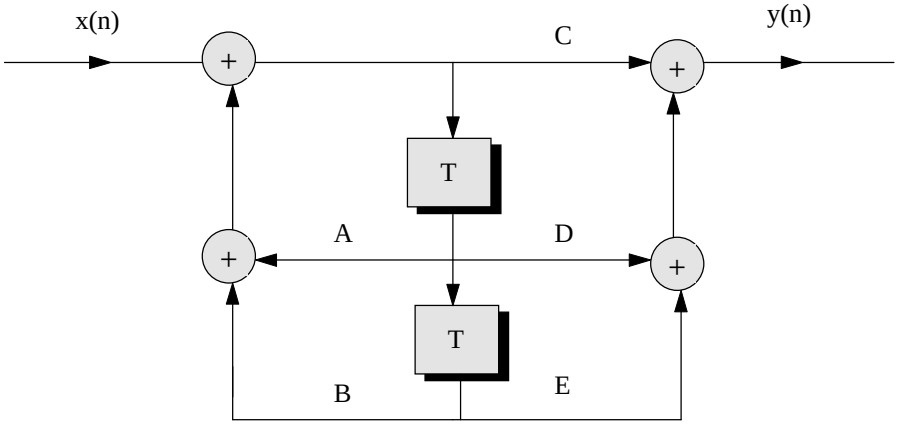
Algumas estruturas comumente utilizadas para a implementação de filtros digitais IIR são discutidas a seguir, inclusive sob o aspecto do seu desempenho em precisão finita.

2.2.1. Estruturas na forma direta

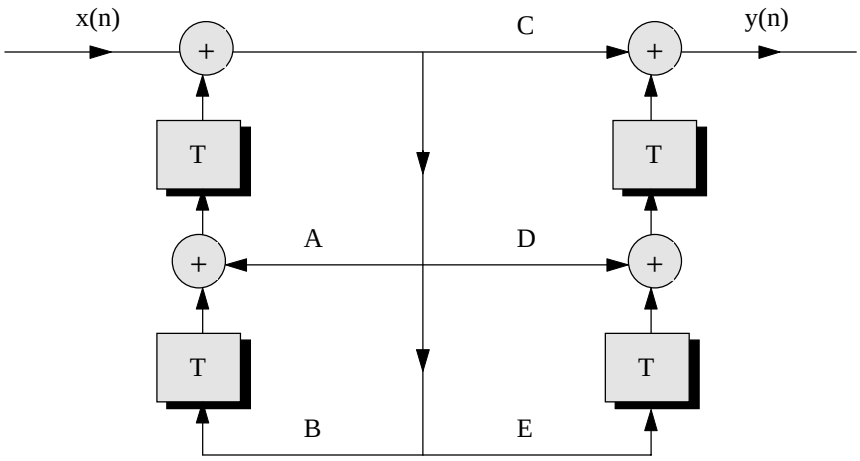
Estruturas na forma direta são obtidas diretamente da equação diferença (2.1). A Figura 2.1 é chamada de forma direta tipo I /Opp89/. Através da conveniente ordenação das partes correspondentes ao numerador e denominador dos polinômios de H(z), e a combinação das linhas de atraso, a forma direta tipo II é obtida. Agora tem-se uma estrutura dita canônica, porque tem o número mínimo de atrasos, multiplicações e somas. Pela aplicação da transposição /Opp89/ duas outras estruturas são obtidas. Elas são chamadas de forma direta tipo I^t e tipo II^t, respectivamente. Todas as seções na forma direta, para o caso de segunda ordem (N = 2 na equação (2.1)), são mostradas na Figura 2.2.



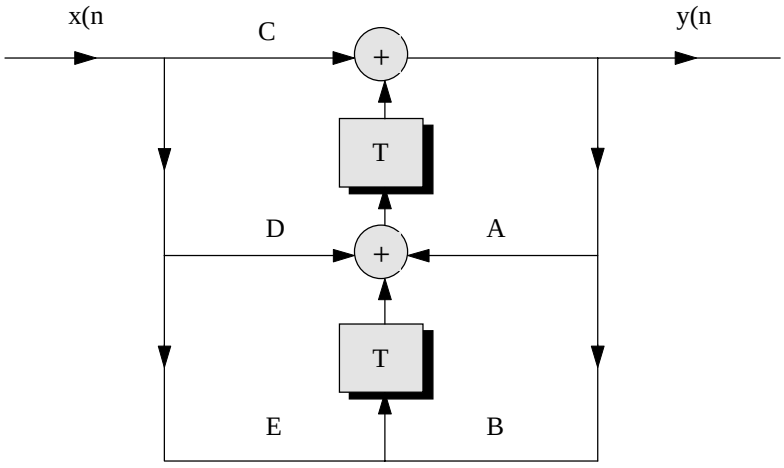
a) Rede na forma direta tipo I.



b) Rede na forma direta tipo II.



c) Rede na forma direta tipo I^t.



d) Rede na forma direta tipo II^t.

Figura 2.2. Seções de segunda ordem na forma direta .

Uma característica marcante das redes na forma direta é que elas são redes de mínima complexidade computacional, pois têm o número mínimo de atrasos, de multiplicadores e de somadores /Opp89/. Porém, elas são excessivamente ruidosas, principalmente, no caso de filtros de banda estreita /Sar90/. Uma solução, nesse caso, é implementar os filtros digitais na forma de conexões em cascata ou em paralelo de blocos de segunda ordem na forma direta.

2.2.2. Estruturas na forma de espaço de estados

A classe das estruturas digitais descritas no espaço de estados tem merecido especial atenção, especialmente pelo seu baixo ruído nos casos de filtros de banda estreita /Sar90/. Porém, devido à sua grande complexidade computacional, estas

estruturas também são normalmente implementadas na forma cascata ou paralela de blocos de segunda ordem /Mul76a/. Assim, o bom desempenho, a nível do ruído de quantização na saída, passa a ser um dos fatores determinantes para a utilização de um determinado bloco de segunda ordem no projeto do filtro cascata ou paralelo /Sim94/.

Por outro lado, a implementação dos filtros digitais com comprimento de palavra finito (por exemplo na implementação em DSP's) pode causar o surgimento de ciclos limite, que são oscilações parasitas totalmente indesejáveis. Portanto, deve-se eliminar tais oscilações. Em /Sim94/ são descritas as técnicas comumente adotadas para a eliminação de ciclos limite à entrada zero, à entrada constante e devidos à "overflow", bem como é comparado o nível de ruído de algumas estruturas representadas no espaço de estados usadas na implementação de filtros digitais. Com isso, fica caracterizado o projeto de filtros digitais de ordem elevada que apresentam baixa complexidade computacional, imunidade a ciclos limite e baixo nível de ruído na saída. As estruturas digitais no espaço de estados são particularmente adequadas para o projeto de filtros digitais com tais características, sobressaindo-se ainda mais quando a banda passante dos filtros é estreita.

2.3. Considerações sobre o comprimento de palavra finito

Vários compiladores desenvolvidos para uso em computadores de uso genérico, ou mesmo processadores de uso específico usando linguagem "assembly", são usados para a implementação de filtros digitais /Opp89, Cha90/. Os resultados armazenados na memória, através de registradores de comprimento de palavra finito, oriundos de operações básicas como somas e multiplicações, vêm acompanhados de erros, acarretados pelo uso destes registradores. Ao representarmos um número através de uma quantidade de bits menor do que o necessário, estamos realizando uma quantização, que pode ter como conseqüência o aparecimento de oscilações parasitas, os ciclos limite, devido ao fato do processo de quantização ser não linear /Sim94/.

2.3.1. Quantização de coeficientes e sinais

Quando um filtro digital é implementado, os coeficientes devem ser quantizados para valores que podem ser representados no sistema numérico usado pelo computador (as principais representações numéricas são as de sinal magnitude, complemento a um e complemento a dois, sendo a última a mais utilizada). Isto faz com que algum erro seja introduzido nos coeficientes e, consequentemente, a resposta em freqüência do filtro seja alterada. O erro entre o número real n_r e sua representação após a quantização, n_q, utilizando-se uma representação com um número finito de bits, é dado por

$$e(n) = n_r - n_q \tag{2.3}$$

onde n_q é um inteiro, múltiplo do passo de quantização q, e (b + 1) bits são usados na quantização. É importante lembrar que o menor valor representável, ou o passo de quantização, vale 2^{-b}.

Existem basicamente três métodos que são usados para quantizar os coeficientes multiplicadores e os resultados dos produtos realizados na computação de um determinado filtro digital: arredondamento, truncamento em complemento a dois e

truncamento em magnitude. Cada um dos três métodos se caracteriza por sua própria esperança e variância estocásticas /Pap84/.

Considerando que o erro e(n) se aproxima do modelo de ruído branco, com distribuição uniforme em amplitude no intervalo [-q/2, q/2] /Sim94, Pap84/, quando os valores são quantizados por arredondamento, temos

$$E[e(n)] = 0 \tag{2.4.a}$$

$$V[e(n)] = \sigma e^2 = q^2/12 = 2^{-2b}/12 \tag{2.4.b}$$

onde E[e(n)] de V[e(n)] denotam o valor esperado e a variância da variável estocástica e(n), respectivamente.

Para o caso de truncamento em complemento a dois tem-se

$$E[e(n)] = -q/2 \tag{2.5.a}$$

$$V[e(n)] = \sigma e^2 = q^2/12 = 2^{-2b}/12 \tag{2.5.b}$$

onde podemos notar que o erro de truncamento em complemento a dois é sempre negativo.

Por fim, para o método de truncamento em magnitude temos

$$E[e(n)] = 0 \tag{2.6.a}$$

$$V[e(n)] = 2^{-2b}/3 \tag{2.6.b}$$

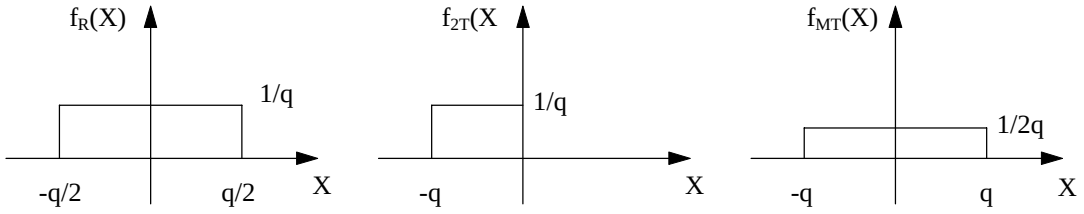
Quando são usados processadores de sinais, o comprimento máximo da palavra é fixo. Porém, o comprimento de palavra dos modernos processadores de sinais (16 bits ou mais) é, na grande maioria das vezes, suficiente para o projeto de filtros práticos.

Alguns procedimentos para otimização discreta têm sido aplicados para se encontrar o melhor conjunto de coeficientes já quantizados /Ren82/, ou seja, a melhor realização. Estes procedimentos podem requerer extensivos cálculos, especialmente quando o filtro é de ordem elevada. Tal problema de aproximação leva a um processo de quantização de coeficientes que depende do método de implementação usado.

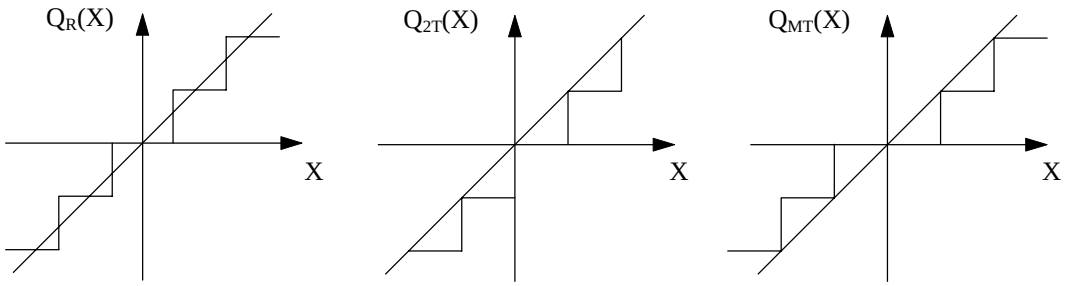
Uma forma de aproximação consiste em considerar a quantização já no procedimento de projeto da função de transferência. O filtro seria projetado de acordo com as especificações e, então, a função de transferência teria seus coeficientes quantizados todos checados. Se a especificação não fosse atendida, um novo projeto, com outras especificações, seria desenvolvido. Alguns programas comerciais para projeto de filtros têm essa possibilidade de aproximação incluindo a quantização /Asp84/, que permite desenvolver um projeto de forma totalmente interativa.

A correspondente função densidade de probabilidade do erro e(n), a característica de quantização e a característica do erro de quantização são mostrados na

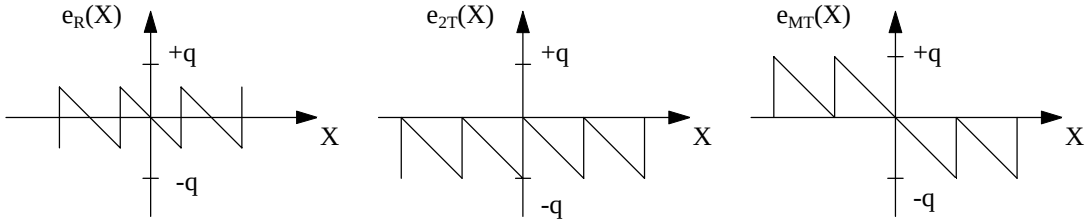
Figura 2.3, para os métodos de arredondamento, truncamento em complemento a dois e truncamento em magnitude.



a) Característica da função densidade de probabilidade do erro.



b) Característica de quantização.



c) Característica de erro de quantização.

Figura 2.3. Características básicas dos métodos de quantização.

Finalmente, deve ser destacado que, ao longo deste trabalho, apenas o truncamento em magnitude será adotado, uma vez que ele é a forma de quantização mais comum em filtragem digital /Vai87, Sim94/. Entretanto, sua implementação é um pouco mais elaborada do que o simples descarte dos bits menos significativos /Tex93, Laa87/.

2.3.2. Ciclos limite e outros efeitos associados a sinais de baixo nível

Alguns efeitos não-lineares podem ser observados, quando da implementação de filtros digitais usando-se DSP's. Isto se deve ao comprimento finito da palavra, que impõe o uso de quantizadores (podendo surgir ciclos limite granulares /Ant93/), ou quando os módulos dos sinais internos excedem a faixa dinâmica adotada para os registradores do DSP (podendo surgir ciclos limite devidos a "overflow" /Ant93/). Entre os efeitos que podem ser observados na saída do filtro podemos destacar:

1. a amplitude do sinal pode ser substancialmente alterada (aumentando ou diminuindo) e a forma de onda pode ser severamente distorcida. O sinal de saída pode também variar enormemente, embora o sinal de entrada seja mantido constante, ou mesmo zerado;

2. o sinal pode ser completamente perdido;

3. o filtro pode ser levado a ter uma oscilação periódica auto-sustentável, o ciclo limite, que aparece quando o sinal de entrada vai para zero ou é mantido constante;

4. o filtro pode ter uma saída oscilatória, mesmo com entrada nula, como conseqüência de um "overflow" em algum registro.

Os ciclos limite são causados pelo processo de quantização na parte recursiva do filtro, constituindo-se de oscilações parasitas que surgem na saída, sustentando-se mesmo que a excitação seja retirada. Observe-se, por exemplo, o bloco recursivo de primeira ordem mostrado na Figura 2.4.

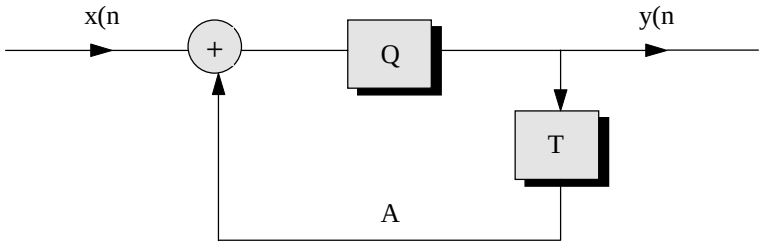


Figura 2.4. Uma seção de primeira ordem de filtro recursivo.

Nos filtros de primeira ordem, somente ciclos limite à entrada zero ou entrada constante (ou constante com sinal alternado) podem ocorrer. Nos filtros de segunda ordem também podem ocorrer outros ciclos limite, por exemplo quando a freqüência está próxima da freqüência ressonante do filtro /Laa87/.

A existência de ciclos limite depende da função de transferência e da estrutura do filtro. A estratégia de projeto mais comum é selecionar estruturas que não apresentem ciclos limite, o que é uma propriedade intrínseca de algumas delas. Em /Sim94/, por exemplo, são discutidas estruturas no espaço de estados que não apresentam ciclos limite.

O método usado na quantização também afeta a existência de ciclos limite /Ant93/. O truncamento em magnitude sempre reduz o módulo do sinal e assim sua energia. Dessa forma, ele é universalmente adotado para a quantização em filtros digitais / Sar90, Sim94/, pois garante que a energia armazenada no sistema irá para zero, quando a entrada é retirada. O mesmo não ocorre no arredondamento e no truncamento em complemento a dois, em que a quantização pode crescer a magnitude do sinal, e assim sua energia.

Para se evitar que sob certas condições oscilações periódicas auto-sustentáveis possam ocorrer, alguns métodos especiais de quantização também têm sido propostos /But75, Kie77/, que chaveiam aleatoriamente entre o arredondamento e o truncamento em magnitude. Porém, eles não serão aqui considerados.

2.3.3. Oscilações devido a overflow

Oscilações por overflow também são causados pelo loop de realimentação dos filtros recursivos. Quando os módulos dos sinais internos excedem a faixa dinâmica dos registros disponíveis, temos um "overflow", o qual, se ocorrer frequentemente em um pequeno período de tempo, pode acarretar severas distorções no sinal de saída do filtro, iniciando oscilações auto-sustentáveis denominadas ciclos limite devidos a "overflow".

Em outras palavras, quando um overflow é causado pelo sinal de entrada, durante as operações de adição, o erro devido ao overflow pode causar um novo overflow e, desta forma, o resultado pode ser uma oscilação que permanece quando o sinal de entrada é levado a zero. Isto é chamado de ciclo limite devido a overflow.

Em alguns processadores, a saturação aritmética é implementada se o acumulador exceder um valor máximo (ou mínimo) /Laa87, Ant93/. Então, o acumulador assume o valor máximo (ou o mínimo) permanecendo, assim, "saturado". Esta é uma forma de se evitar a ocorrência de oscilações em caso de "overflow" /Sin85/. Porém, se a operação de saturação for realizada também nos resultados intermediários, pode-se causar uma distorção desnecessária. Normalmente ela só é aplicada ao resultado da adição da última parcela /Laa87/.

2.4. As estruturas implementadas neste trabalho

Uma equação diferença de ordem N pode ser escrita como um conjunto de N equações de primeira ordem /Ant93/. Elas podem ser expressas como equações de estado, na forma matricial, através de:

$$\mathbf{S}(n+1) = \mathbf{A}\mathbf{S}(n) + \mathbf{b}\mathbf{x}(n) \qquad (2.7.a)$$

$$y(n) = \mathbf{c}^t\mathbf{S}(n) + d\mathbf{x}(n) \qquad (2.7.b)$$

onde x(n) e y(n) (escalares) são, respectivamente, a entrada e a saída, **S**(n) é um vetor de dimensão N, constituído das variáveis de estado (as saídas dos atrasos), **A** é uma matriz NxN, **b** e **c** são vetores de dimensão N e d é um escalar. Os elementos que constituem **A**, **b**, **c** e d são os coeficientes do filtro. Pode-se observar que o número de coeficientes aumenta rapidamente com a ordem N do filtro (são $(N + 1)^2$ coeficientes). Qualquer estrutura, mesmo na forma direta, pode ser descrita usando-se a formulação no espaço de estados. A diferença está nos valores de **A**, **b**, **c** e d. Um diagrama de blocos descrevendo esta formulação é mostrado na Figura 2.5.

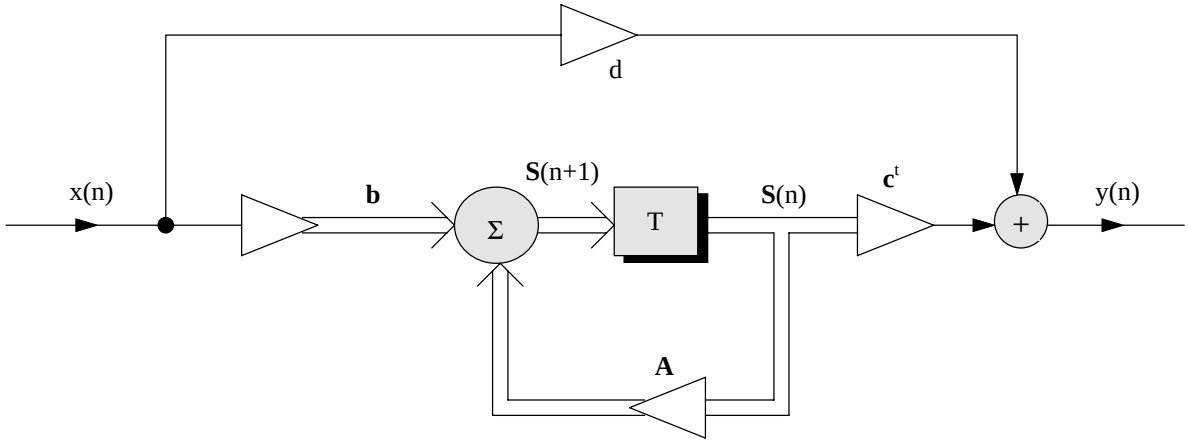


Figura 2.5. Diagrama de blocos para a formulação geral em espaço de estados /Laa87/.

A estrutura detalhada do filtro de segunda ordem no espaço de estados é mostrada na Figura 2.6.

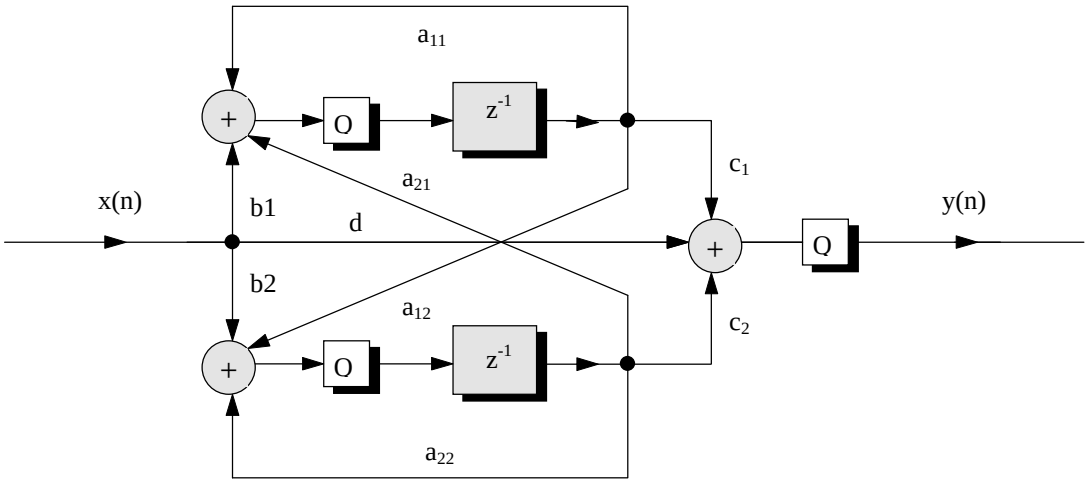


Figura 2.6. Seção de segunda ordem no espaço de estados.

Observe-se, em tal figura, que a quantização é feita após a soma, o que exige precisão dupla dos somadores, do registro que armazena o resultado dos produtos e do acumulador que armazena o resultado das somas. Note-se que isto é compatível com a utilização de DSP's /Cha90, Tex93/.

A função de transferência do filtro é

$$H(z) = d + \mathbf{c}^t(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{b} \qquad (2.8)$$

Nos casos de filtros de segunda ordem tem-se

$$H(z) = d + \frac{n_1 z^{-1} + n_2 z^{-2}}{1 - (a_{11} + a_{22})z^{-1} + (a_{11}a_{22} - a_{12}a_{21})z^{-2}} = \frac{N(z)}{D(z)} \qquad (2.9.a)$$

onde

$$n_1 = c_1 b_1 + c_2 b_2 \tag{2.9.b}$$

$$n_2 = c_1(a_{12} b_2 - a_{22} b_1) + c_2(a_{21} b_1 - a_{11} b_2) \tag{2.9.c}$$

e daí o numerador N(z) é facilmente obtido como

$$N(z) = d\left\{1 + \left[\frac{n_1}{d} - (a_{11} - a_{22})\right] z^{-1} + \left(\frac{n_2}{d} + a_{11} a_{22} - a_{12} a_{21}\right) z^{-2}\right\} \tag{2.10}$$

A principal vantagem desta estrutura é que existem infinitas soluções para os coeficientes do filtro, dentre as quais um conjunto particular de coeficientes que garantem mínimo ruído, bem como uma pequena sensibilidade à variação dos coeficientes /Jac79/. Inclusive, essas duas propriedades são correlacionadas, ou seja, um baixo ruído e uma baixa sensibilidade são usualmente obtidas ao mesmo tempo /Jac79/, o que permite definir como critérios essenciais de projeto apenas a imunidade a ciclos limite, se for o caso, e o baixo ruído /Sar90/.

Como visto anteriormente, blocos básicos de segunda ordem organizados na forma cascata ou paralela são normalmente usados na implementação de filtros digitais de ordem elevada. A redução da sensibilidade da rede, assim como do ruído na saída dos filtros são, entre outras, as principais vantagens desta forma de implementação. Porém, o ruído na saída dos filtros implementados na forma cascata ou paralela de seções diretas de segunda ordem, nos casos de filtros digitais de banda estreita, ainda é elevado /Jac70/.

Para resolver este problema procurou-se, então, implementar os filtros sob a forma de estruturas no espaço de estados, sintetizando-se uma estrutura em que a variância relativa do ruído na saída não só é mínima como também é invariante com relação à largura da banda passante /Mul76a, Mul76b/. Entretanto, dada a grande quantidade de produtos necessários para computar tal estrutura, usa-se, atualmente, sintetizar estruturas na forma cascata ou paralela em que cada bloco de segunda ordem é uma estrutura descrita no espaço de estados de mínimo ruído /Jac79/, obtendo-se um bom compromisso entre baixa complexidade computacional e baixo ruído /Mul76b/.

Inúmeros esforços tem sido desenvolvidos para se propor estruturas de segunda ordem descritas no espaço de estados que apresentem bom desempenho a nível de ruído na sua saída, além de outras características de interesse, como imunidade a ciclos limite /Sim94, Bom85, Bom89, Din86, Jac79, Sar92/.

A estrutura de segunda ordem de mínimo ruído /Jac79/, a estrutura quase ótima imune a ciclos limite /Sar92/ e duas estruturas de segunda ordem imunes a ciclos limite /Din86/ são usadas como referência para a plataforma implementada neste trabalho. Assim, esta seção destaca suas propriedades e a análise de sua complexidade computacional, muito importante no que se refere à sua implementação.

Considere-se, agora, o escalamento das estruturas no espaço de estados. Os pontos de escalamento são os nós correspondentes às variáveis de estado. Os quantizadores são colocados nos nós de entrada dos atrasos, e também na saída. Os vetores de funções de transferência considerados no escalamento e no cálculo do ruído são, respectivamente

$$\mathbf{F}(z) = [F_1(z)F_2(z) \,..\, F_N(z)] = (z\mathbf{I} - \mathbf{A})^{-1}\,\mathbf{b} \tag{2.11}$$

$$\mathbf{G}(z) = [G_1(z)G_2(z) \,..\, G_N(z)] = (z\mathbf{I} - \mathbf{A})^{-1}\,\mathbf{c} \tag{2.12}$$

No caso de segunda ordem, as funções de transferência são (veja também /Laa87/):

$$F_1(z) = z^{-1}[b_1 + (a_{12}b_2 - a_{22}b_1)z^{-1}]/D(z) \tag{2.13.a}$$

$$F_2(z) = z^{-1}[b_2 + (a_{21}b_1 - a_{11}b_2)z^{-1}]/D(z) \tag{2.13.b}$$

$$G_1(z) = z^{-1}[c_1 + (a_{21}c_2 - a_{22}c_1)z^{-1}]/D(z) \tag{2.13.c}$$

$$G_2(z) = z^{-1}[c_2 + (a_{12}c_1 - a_{11}c_2)z^{-1}]/D(z) \tag{2.13.d}$$

Uma transformação de similaridade que resulta no filtro escalado é, então, dada pela matriz diagonal **T** definida por

$$\mathrm{T} = \mathrm{diag}\left(\|F_1\|_p^{-1}\,\|F_2\|_p^{-1}\,..\|F_N\|_p^{-1}\right) \tag{2.13.e}$$

onde p = 2 para escalamento L_2 e p = ∞ para escalamento L_∞. Tal transformação de escalamento afeta os coeficientes do filtro da seguinte forma:

$$\bar{\mathbf{A}} = \mathbf{TAT}^{-1}, \qquad \bar{\mathbf{b}} = \mathbf{Tb}, \qquad \bar{\mathbf{c}}^t = \mathbf{c}^t\mathbf{T}^{-1}, \qquad \bar{d} = d \tag{2.13.f}$$

É facilmente verificado que a função de transferência não é alterada por uma transformação deste tipo. No caso de blocos de segunda ordem, **T** = diag(g_1 g_2) e os seguintes coeficientes são alterados:

$$\bar{a}_{12} = \frac{g_1}{g_2}a_{12} \qquad \bar{a}_{21} = \frac{g_2}{g_1}a_{21} \tag{2.14.a}$$

$$\bar{b}_1 = g_1 b_1 \qquad \bar{b}_2 = g_2 b_2 \qquad \bar{c}_1 = \frac{c_1}{g_1} \qquad \bar{c}_2 = \frac{c_2}{g_2} \tag{2.14.b}$$

onde

$$g_1 = \frac{1}{\|F_1\|_2} \quad \text{e} \quad g_2 = \frac{1}{g_1\|H_1F_2\|_2} \tag{2.14.c}$$

2.4.1. A estrutura de mínimo ruído

O projeto da estrutura de mínimo ruído de ordem N foi originalmente formulado por Mullis e Roberts /Mul76a/, considerando o escalamento quadrático L_2. Mais tarde Jackson et. al. /Jac79/ propuseram fórmulas explícitas para projeto de filtros de segunda ordem no espaço de estados de mínimo ruído, também com escalamento quadrático L_2, que é o único caso em que se pode minimizar o ruído. Para o caso de escalamento L_∞, pode-se obter baixo ruído, mas não o mínimo global /Jac79, Din86, Bom89/.

Em /Mul76a, Jac79/ é mostrado que o escalamento L_2 é ótimo quando todas as fontes de ruído tiverem igual contribuição para o ruído na saída, ou seja, para o caso de ordem 2

$$\|\bar{G}_1\|_2^2 = \|\bar{G}_2\|_2^2 \qquad \forall\ ij \qquad (2.15.a)$$

onde

$$\bar{G}_i(z) = \|F_i\|_2 G_i(z),\ i = 1, 2, \ldots \qquad (2.15.b)$$

sendo

$$\|F_1(z)\|_2 = \|F_2(z)\|_2 = 1 \qquad (2.15.c)$$

onde $\bar{G}_i(z)$ é chamada a função de transferência escalada do ruído. No caso de blocos de segunda ordem, tem-se que as condições acima resultam em

$$a_{11} = a_{22} \qquad c_1 b_1 = c_2 b_2 \qquad G_1(z) = kF_2(z) \qquad G_2(z) = kF_1(z) \qquad (2.16)$$

onde k é uma constante.

A Figura 2.7 descreve a estrutura no espaço de estados de mínimo ruído, de segunda ordem, incluindo as operações não lineares de quantização, realizadas nas variáveis de estado e na saída, ou seja, após as somas.

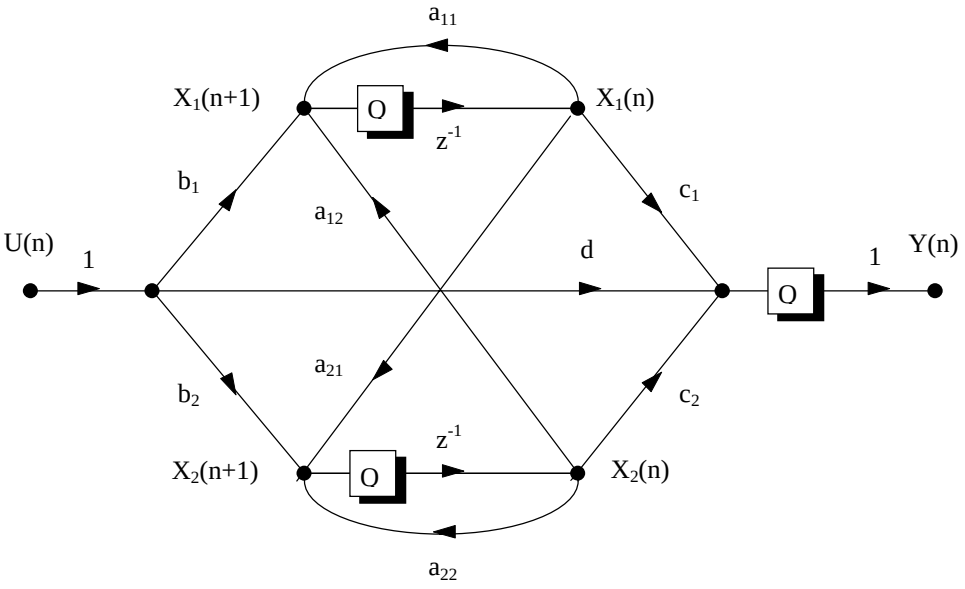


Figura 2.7. A estrutura de mínimo ruído.

A estrutura de mínimo ruído é, portanto, passível de implementação em DSP, e por isso de interesse para esse trabalho. Como as quantizações ocorrem depois de somas, Q também inclui o tratamento de overflow.

Dado que

$$H(z) = d + \frac{n_1 z^{-1} + n_2 z^{-2}}{1 - Az^{-1} - Bz^{-2}} \tag{2.17a}$$

e considerando que uma seção de segunda ordem também é descrita por

$$H(z) = \frac{C + Dz^{-1} + Ez^{-2}}{1 - Az^{-1} - Bz^{-2}} = \frac{N(z)}{D(z)} \tag{2.17b}$$

obtém-se os coeficientes

$$d = C \qquad n_1 = CA + D \qquad n_2 = CB + E \tag{2.18}$$

Os coeficientes da rede ótima não escalada são, então, obtidos pelas seguintes fórmulas:

$$a_{11} = a_{22} = \frac{A}{2} \qquad a_{12} = \frac{(1+n_2)(K_1 \pm K_2)}{n_1^2} \qquad a_{21} = \frac{K_1 \mu K_2}{1+n_2} \tag{2.19.a}$$

$$b_1 = \frac{1+n_2}{2} \qquad b_2 = \frac{n_1}{2} \qquad c_1 = \frac{n_1}{1+n_2} \qquad c_2 = 1 \tag{2.19.b}$$

onde

$$K_1 = n_2 + \frac{An_1}{2} \qquad K_2 = \sqrt{n_2^2 + n_1 n_2 A - n_1^2 B} \tag{2.19.c}$$

(As duas possíveis soluções para a_{12} e a_{21} vêm da simetria da estrutura. Note-se que o produto que ocorre na função de transferência é o mesmo em ambos os casos.) Usando-se as equações (2.13), a rede escalada é então obtida /Jac79/.

O resultado obtido em /Vai87/ permite verificar rapidamente que a estrutura de mínimo ruído é imune a ciclos limite, no caso de entrada zero, para quantização por truncamento em magnitude realizada nas variáveis de estado. Uma outra propriedade muito importante da rede de mínimo ruído é que oscilações devido a overflow com entrada zero não ocorrem quando usamos a aritmética de ponto fixo com representação em complemento a dois acompanhada ou não da saturação em caso de overflow /Sin85, Mil78/.

2.4.2. A estrutura quase ótima

Esta estrutura é derivada da rede ótima, com o objetivo de gerar uma rede imune a qualquer tipo de ciclos limite, aproveitando que a rede ótima original já é imune a ciclos limite nos casos de entrada zero e de overflow /Sar92/. A Figura 2.8 ilustra a estrutura quase ótima sem ciclos limite. Ela é obtida a partir da Figura 2.7 aplicando-se a técnica básica de eliminação de ciclos limite no caso de entrada constante /Din86/. Observe-se, inclusive, que alguns coeficientes da rede ótima ainda se mantêm, como é o caso de a_{11op}, a_{22op} e d_{op}. Para os detalhes de sua síntese, ver /Sar92/.

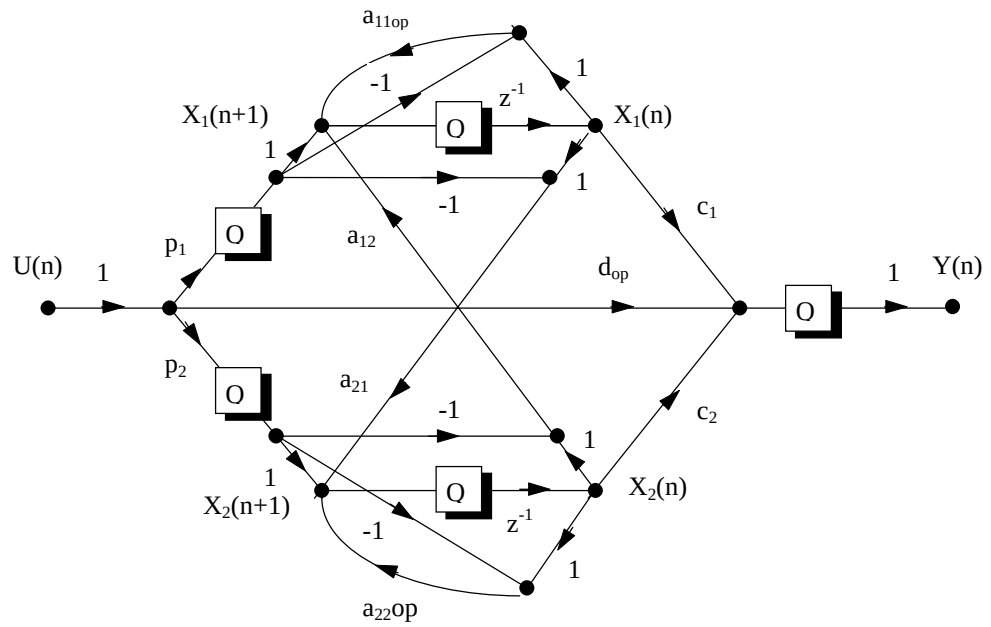


Figura 2.8. A estrutura quase ótima livre de ciclos limite.

A imunidade a ciclos limite, no caso de entrada zero e devidos a “overflow”, está garantida pela imunidade a ciclos limite no caso da entrada zero da rede ótima /Vai87/, haja vista que a estrutura recursiva mostrada na Figura 2.8 é a mesma da Figura 2.7, a menos do escalamento. Observe-se, também, que tal estrutura é computacionalmente mais complexa que a seção ótima, mas ainda é adequada para a implementação em DSP.

2.4.3. As estruturas sem ciclos limite tipos I e III

Nas Figuras 2.9 e 2.10 são mostradas duas outras estruturas de segunda ordem também imunes a qualquer tipo de ciclos limite, chamadas estruturas sem ciclos limite tipo I e III, respectivamente /Din86/.

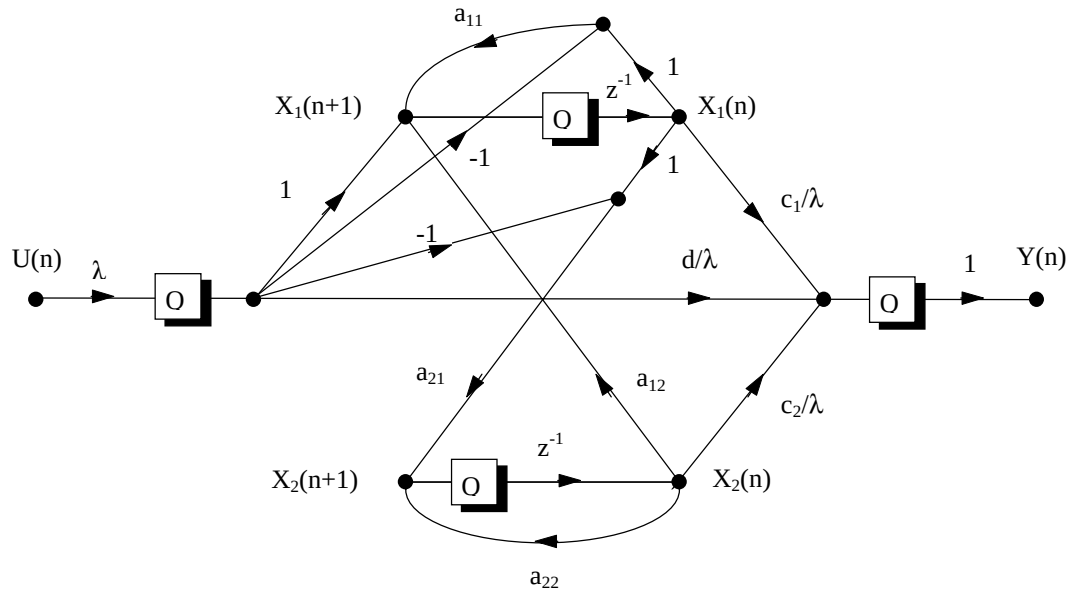


Figura 2.9. Estrutura de segunda ordem sem ciclos limite tipo I.

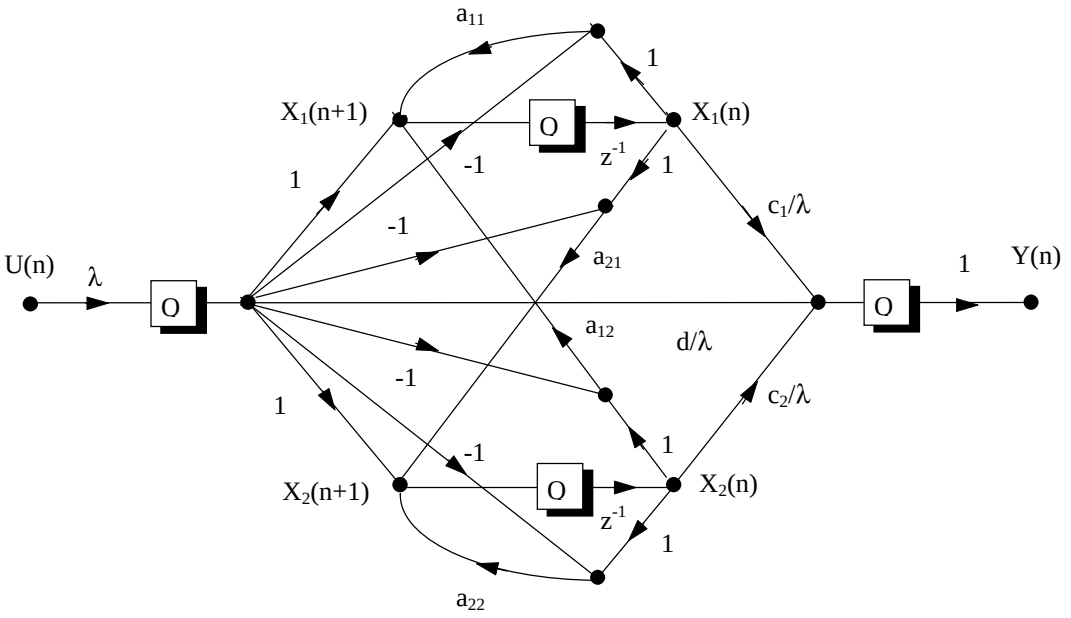


Figura 2.10. Estrutura de segunda ordem sem ciclos limite tipo III.

Os detalhes de sua síntese, tanto para a conexão em cascata como para a conexão em paralelo, são abordados pelos autores que as propuseram. Sob a ótica do presente trabalho, sua compatibilidade para implementação em DSP é um fator a ser destacado, assim como o fato de que elas são menos complexas que as seções das Figuras 2.7 e 2.8.

2.4.4. A complexidade computacional das estruturas

A seguir é discutida a complexidade computacional inerente às estruturas de segunda ordem de mínimo ruído, quase ótima e sem ciclos limite dos tipos I e III. A estrutura da Figura 2.8 difere daquela da Figura 2.7, sob o ponto de vista de quantização, pela necessidade de dois quantizadores para o produto **P**U[n]. Já as estruturas das

Figuras 2.9 e 2.10 têm o multiplicador genérico adicional λ, e consequentemente um quantizador para o produto λU[n]. Logo, por cada seção, prevalece o número de sete multiplicadores, no caso das Figuras 2.9 e 2.10, e oito por cada seção das Figuras 2.7 e 2.8.

As Tabelas 2.1 e 2.2 dão uma idéia da complexidade de tais seções. Na tabela 2.1 é mostrado o número de somadores, multiplicadores genéricos e quantizadores necessários para implementar uma estrutura paralela de N blocos como os das Figuras 2.7, 2.8, 2.9 e 2.10. Deve-se notar que um único coeficiente d é utilizado, ao invés de um por seção, de acordo com a equação (2.8). Também é usado um multiplicador λ por cada seção das Figuras 2.9 e 2.10.

TIPO DE ESTRUTURA	Número de Quantizadores	Número de Multiplicadores	Número de Somadores
Mínimo ruído	**2N + 1**	**8N + 1**	**6N**
Quase ótima	**4N + 1**	**8N + 1**	**10N**
SCL tipo I	**3N + 1**	**7N + 1**	**7N**
SCL tipo III	**3N + 1**	**7N + 1**	**10N**

Tabela 2.1. Implementação de um filtro paralelo de N blocos de segunda ordem.

Já a tabela 2.2 indica o hardware necessário para implementar um filtro na forma cascata de N blocos, usando as seções acima. Aqui, o multiplicador λ na entrada de cada seção das Figuras 2.9 e 2.10 é considerado na seção anterior, exceto a primeira. Assim, um único λ é usado, enquanto é mantido um d diferente para cada seção, em todos os casos. Agora, são 9 multiplicadores por cada seção das Figuras 2.7 e 2.8, e 7 multiplicadores por cada seção das Figura 2.9 e 2.10, e mais um único λ, nestes dois últimos casos. No que se refere a quantizadores, são 3 por cada seção das Figuras 2.7, 2.9 e 2.10, e 5 por cada seção da Figura 2.8.

TIPO DE ESTRUTURA	Número de Quantizadores	Número de Multiplicadores	Número de Somadores
Mínimo ruído	**3N**	**9N**	**6N**
Quase ótima	**5N**	**9N**	**10N**
SCL tipo I	**3N + 1**	**7N + 1**	**7N**
SCL tipo III	**3N + 1**	**7N + 1**	**10N**

Tabela 2.2. Implementação de um filtro cascata de N blocos de segunda ordem.

Para os objetivos deste trabalho, em que apenas a implementação em DSP é utilizada para a realização de filtros digitais, as quatro estruturas das Figuras 2.7, 2.8, 2.9 e 2.10 são interessantes, dado que a diferença entre elas, em termos das operações necessárias, não são impeditivas, dada a velocidade de processamento do DSP utilizado /Cha90, Tex93/. Por outro lado, sua imunidade a ciclos limite e o baixo ruído na sua saída /Jac79, Din86, Sar92, Sim94/ as tornam atrativas, e são a razão de sua escolha, no

presente trabalho, como blocos básicos para a implementação de filtros digitais nas formas cascata e paralela. Deve ser enfatizado que devido a modularidade da plataforma, outras estruturas também podem ser incluídas com certa facilidade. Nos Capítulos 3 e 5, principalmente, tal implementação é devidamente abordada.

CAPÍTULO 3

Implementação de Filtros Digitais em Processadores Digitais de Sinais

Neste capítulo, é discutida a implementação de filtros digitais usando o processador digital de sinais TMS320C25, assim como algumas características desse processador. A implementação da multiplicação e da quantização por truncamento em magnitude também serão examinadas, assim como será caracterizado o programa correspondente a um filtro na forma paralela de blocos de segunda ordem. Finalmente, é feita uma discussão sobre desempenho, no que se refere à implementação das estruturas de filtros recursivos em processadores digitais de sinais.

3.1. Os processadores de sinais modernos

O primeiro microprocessador projetado especialmente para aplicações no processamento de sinais data de 1979. Era o chip Intel 2920 /Laa87/, um pequeno processador dotado de conversores A/D e D/A de 9 bits. Ele possibilitou a implementação de algoritmos para processamento digital de sinais de forma bastante simplificada. Mesmo assim, a implementação de tais algoritmos ainda consumia muito tempo e era altamente dispendiosa.

Os algoritmos para processamento de sinais, tais como módulos de FFT, filtros IIR ou filtros FIR, são estruturados de tal forma que sejam independentes dos dados e façam uso abundante das operações aritméticas básicas de adição, multiplicação e deslocamento. Nos microprocessadores convencionais, como por exemplo os microprocessadores de uso genérico (microcomputadores), a multiplicação tipicamente é uma operação demorada, consumindo muitos ciclos de máquina, enquanto que nos processadores digitais de sinais (que são especialmente projetados para tanto) existem módulos separados para a multiplicação, que são mais rápidos, permitindo assim realizar a multiplicação no mesmo tempo que a operação de adição /Tex93, Cha90/. Adicionalmente, tanto os ciclos de instrução como o acesso à memória de dados são bem mais rápidos.

Os processadores da Intel, tais como o Intel 2920, foram uma solução compacta para pequenos sistemas por causa da inclusão em hardware dos conversores A/D e D/A, mas sua pequena capacidade de memória e a lentidão na execução de um ciclo de instrução (aproximadamente 400 ns) tornou este processador insuficiente para muitas aplicações /Laa87/.

O primeiro processador de sinais largamente usado foi o NEC μPD 7720, lançado em meados de 1980 /Laa87/. Suas memórias de programa e de dados eram muito maiores e ele possuía um ciclo de instrução bem menor (aproximadamente 250 ns). Ele também tinha um multiplicador rápido, implementado em hardware na forma de 16x16 bits, que executava uma instrução de multiplicação em apenas um ciclo de máquina. Um programa montador e um outro emulador de hardware também foram desenvolvidos para suportar aplicações usando esse processador.

3.1.1. O processador de sinais TMS32010 /Tex83, Laa87/

O processador de sinais da Texas Instruments Inc. TMS32010, foi lançado em 1982. Ele era mais eficiente que o processador da NEC, possuindo um ciclo de instrução de apenas 200ns. O TMS32010 usava uma arquitetura do tipo Harvard, com barramentos separados para dados e programa, o que permitia um processamento completamente paralelo.

O processador TMS32010 trabalhava com 16 bits de comprimento de palavra. Não continha memória de programa intra-chip (somente a versão TMS320M10 tinha 1536 palavras de memória ROM de programa interna ao chip), mas uma memória de programa externa de até 4k palavras podia ser usada. Existiam somente 144 palavras de memória de dados (RAM) interna para dados e coeficientes, mas os dados também podiam ser armazenados na memória de programa. Muitas das suas instruções gastavam apenas um ciclo de máquina, incluindo aí a multiplicação de 16x16 bits. Instruções de I/O e de desvio gastavam 2 ciclos, enquanto instruções para transferência de dados entre as memórias de dados e de programa gastavam 3 ciclos.

A unidade lógica e aritmética (ULA) e o acumulador eram de 32 bits. O "barrel shifter" deslocava 0, 1, 2, ..., 16 bits para a esquerda, quando o dado era carregado da memória de dados para o acumulador. Também era possível deslocar um valor de 0, 1 ou 4 bits para a esquerda, quando de sua armazenagem na memória de dados. O multiplicador multiplicava uma palavra armazenada no registro T pela palavra endereçada na memória de dados ou, no caso de multiplicação imediata, com a constante desejada, de 13 bits (podia-se endereçar até 4096 palavras na memória de programa externa ao chip). O resultado era armazenado no registro P, de 32 bits, que por sua vez podia ser armazenado ou adicionado ao acumulador via uma operação paralela.

Algumas instruções, como as de multiplicação, combinavam operações em paralelo. Elas eram especialmente otimizadas para a implementação rápida de adições, multiplicações e transferência de dados, que são as operações básicas necessárias nos algoritmos de processamento de sinais.

O overflow podia ser prevenido pelo uso do modo overflow, no qual o maior número positivo, ou menor número negativo, representados em complemento a dois, era carregado no acumulador, no caso de overflow ou underflow, respectivamente.

As ferramentas de suporte aos desenvolvedores para o TMS32010 consistiam de um programa montador, um simulador e um emulador de hardware, produzidos pela própria Texas Instruments Inc. /Tex83, Laa87/.

Durante 4 anos seguidos o TMS32010 foi largamente usado. Ele podia ser considerado "de facto" um padrão na indústria de comunicações. Alguns aplicativos comerciais foram desenvolvidos, incluindo programas de projeto de filtros que "montavam" o programa do filtro diretamente para o processador TMS32010 /Asp84/.

3.1.2. O processador de sinais TMS320C25 /Tex93/

O chip TMS320C25, um processador digital de sinais da família TMS320C2X, lançado em meados de 1986 pela Texas Instruments Inc., combina alto desempenho com algumas características especiais requeridas nas aplicações de processamento digital de sinais.

3.1.2.1. Características do TMS320C25

A arquitetura da família TMS320C2X, constituída pelas versões TMS320C25/E25, TMS320C25-33/C25-50 e o TMS320C26, é uma versão melhorada da primeira geração, constituída pelos processadores TMS32010 e TMS320M10. Ao longo deste capítulo serão apresentados detalhes das características e da arquitetura do TMS320C25.

Como principais características desse processador podemos citar:

- ciclo de instrução de 100 ns, usando clock de 40 MHz;
- memória de programa interna de 4096 palavras, e, externa com até 64 k palavras;
- 544 palavras de memória de dados RAM interna ao chip, e até 64 k externa ao chip ;
- barramento de dados e de programa de 16 bits;
- ULA e acumulador de 32 bits;
- multiplicação de 16 x16 bits em apenas um ciclo de máquina;
- 16 portas de entrada e saída;
- 8 níveis de pilha;
- aritmética de ponto fixo com representação em complemento a dois;
- tecnologia CMOS;
- 5 volts de tensão de alimentação;
- 68 pinos, no formato PGA.

A família de processadores TMS320C2X possui um conjunto de instruções que permite suportar um grande campo de aplicações onde se necessite de computação intensiva e alta velocidade, tais como filtragem digital, processamento de imagem, análise espectral, modems de alta velocidade, reconhecimento de voz, etc. Inclusive, algumas instruções são otimizadas para executar tarefas complexas em um único ciclo de máquina.

Estes processadores não se restringem somente à área de processamento de sinais, podendo ser utilizados em aplicações de propósito geral onde se necessite de uma unidade lógica e aritmética (ULA) e um multiplicador de alto desempenho.

A Figura 3.1 mostra um diagrama de blocos representando a arquitetura interna do TMS320C25.

Figura 3.1. Diagrama em blocos representativo da arquitetura do TMS320C25.

A arquitetura "Harvard", adotada pela maioria dos processadores digitais de sinais, caracteriza-se por ter espaços de memória e barramentos independentes para dados e programas, permitindo um alto grau de paralelismo nas operações, pois os ciclos de busca e de execução de uma instrução podem ocorrer simultaneamente. Este tipo de arquitetura aumenta consideravelmente a capacidade de processamento, mas, por outro lado, tem a desvantagem de possuir a memória de programa totalmente separada da memória de dados.

Por possuir uma interface entre os barramentos de dados e de programas, pode-se dizer que a arquitetura do TMS320C25 é uma arquitetura "Harvard" modificada, pois possibilita a troca de dados entre as memórias de dados e de programas, permitindo que constantes possam ser armazenadas na memória de programa, o que na arquitetura “Harvard” convencional não é possível.

A superposição dos ciclos de busca e execução no TMS320C25 pode ser vista na Figura 3.2.

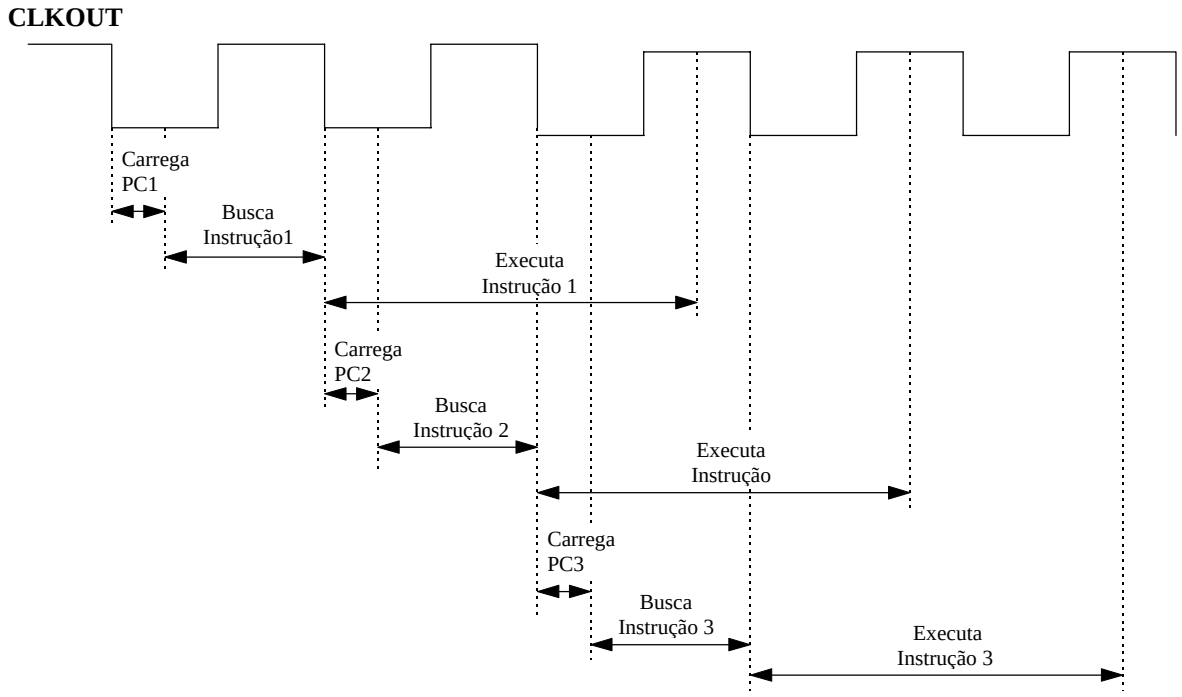


Figura 3.2. Diagrama dos ciclos de busca e execução do TMS320C25.

Na descida do sinal "CLKOUT”, ou seja, ao final do primeiro ciclo, o contador de programas (Program Counter - PC) é carregado com a instrução "Carrega PC2". Inicia-se então a busca da segunda instrução "Busca Instrução 2”, enquanto a instrução atual "Instrução 1" é decodificada pelo processador e executada. A segunda instrução "Instrução 2", ao ser encontrada pelo processador, é decodificada e tem sua execução iniciada antes do término da execução da instrução "Instrução 1", como pode ser visto pela figura. O processador TMS320C25 ainda é capaz de iniciar o processo de busca de uma terceira instrução, "Busca Instrução 3", em paralelo com a execução da instrução atual "Executa Instrução 2" e da instrução anterior "Executa Instrução 1". Esta capacidade do processador só é possível devido às características da arquitetura, que possibilita um alto grau de paralelismo.

A utilização de "hardware" para implementar funções que outros microprocessadores executam através de "software" é uma outra característica que também diferencia a arquitetura dos microprocessadores da família TMS320. O multiplicador paralelo que é implementado por "hardware", executando uma multiplicação de 16 x 16 bits em um único ciclo de máquina (100 ns), é um exemplo típico dessa característica.

3.1.2.2. Elementos aritméticos

A Unidade Lógica e Aritmética (ULA), o acumulador (ACC), o multiplicador e os "shifters" constituem basicamente os quatro elementos aritméticos do TMS320C25. Todos eles operam com aritmética de ponto fixo e representação numérica em complemento a dois.

3.1.2.2.1. Unidade lógica e aritmética

Na Unidade Lógica e Aritmética (ULA) do TMS320C25, operando com 32 bits, são executadas todas as operações de soma, subtração e operações lógicas “XOR”, “AND” e “OR”. O acumulador é sempre o destino da ULA e o operando primário.

A possibilidade de operar em "Overflow Mode" (OVM), constitui um das características mais importantes da ULA. Na ocorrência de um "overflow", o acumulador é carregado com o maior valor positivo ou menor valor negativo que a ULA puder representar. O flag OVM indica o modo de operação da ULA, sendo afetado pela execução das instruções SOVM “Set Overflow Mode” e ROVM “Reset Overflow Mode”. Se for executada a instrução SOVM, o flag OVM é carregado com o estado lógico 1 (um), habilitando desta forma a ULA para operar no "Overflow Mode". Se o inverso ocorrer, ou seja, se for executada a instrução ROVM, o flag OVM é carregado com o estado lógico 0 (zero), desabilitando a ULA a operar no "Overflow Mode".

3.1.2.2.2. Acumulador

Os dados oriundos da ULA são armazenados em um registro de 32 bits, denominado acumulador (ACC). A ULA também pode utilizar o acumulador como seu operando primário, como citado na seção anterior. Duas palavras de 16 bits dividem logicamente o acumulador, podendo ser manipuladas independentemente por várias instruções do TMS320C25. A parte alta do acumulador é formada pelos bits 31 a 16, e a parte baixa é formada pelos bits 15 a 0.

O flag denominado “OV” indica o status do acumulador após uma operação aritmética. Se ocorrer um "overflow" no ACC, esse flag irá para o nível lógico 1 (um). Somente a execução da instrução BV “Branch on Overflow” poderá desatibilitar o flag “OV” (OV = 0).

3.1.2.2.3. Multiplicador

O multiplicador paralelo da família TMS320C2X, de 16 x 16 bits, implementado em "hardware", consiste basicamente de três unidades: um registro T, um registro P e o

"array" multiplicador. Responsável pelo armazenamento do multiplicando, o registro T possui 16 bits. Já o registro P, responsável pelo armazenamento do produto, possui 32 bits.

Deve-se carregar primeiro o multiplicando, para em seguida executar a operação de multiplicação. A carga do multiplicando no registro T pode ser feita através das instruções LT “Load T Register”, LTA “Load T Register and Accumulate Previous Product” e LTD “Load T Register, Accumulate Previous Product, and Move Data Memory”. As instruções MPY “Multiply” e MPYK “Multply Immediate” permitem executar a operação de multiplicação. O produto é armazenado no registro P logo após a execução de uma instrução MPY ou MPYK. Esse produto pode então ser somado ou subtraído do acumulador (ACC), ou somente armazenado. Para tanto, o TMS320C25 possui as instruções APAC “Add P register to ACC”, SPAC “Subtract P Register from ACC”, PAC “Load ACC from P register”, além das já citadas LTA e LTD.

3.1.2.2.4. Shifters

O "barrel shifter" e o "parallel shifter" constituem os dois "shifters" usados para manipulação de dados no TMS320C25. O “barrel shifter” opera na entrada da ULA, ou seja, nos dados provenientes da memória de dados. Já o “parallel shifter”, localizado na saída do acumulador, opera nos dados que vão ser armazenados na memória de dados.

O primeiro “shifter” permite que um dado proveniente da memória de dados seja deslocado para a esquerda de 0 a 15 bits, para posteriormente ser carregado, subtraído ou somado ao conteúdo do acumulador. Neste tipo de operação os bits menos significativos são preenchidos com o mesmo valor do bit de sinal.

Já o segundo “shifter” permite que o conteúdo do acumulador, com seus 32 bits, possa ser deslocado para a esquerda de 0, 1 ou 4 bits antes que os 16 bits mais significativos do acumulador sejam armazenados na memória de dados. A instrução “store high order accumulator”, SACH, é única instrução que ativa esse recurso. Nas operações de multiplicação para escalamento do resultado, por exemplo, o “parallel shifter" constitui num recurso bastante eficaz.

3.1.2.3. Registros auxiliares

O TMS320C25 possui oito registros auxiliares, AR0 a AR7, todos de 16 bits, que podem ter até três funções distintas: armazenamento temporário, endereçamento da memória de dados e controle de "loop".

Os oito bits menos significativos dos registros auxiliares são o endereço da memória de dados, quando utilizado o modo de endereçamento indireto. Pode-se ainda incrementar ou decrementar o conteúdo dos registros auxiliares automaticamente, nesse modo de endereçamento.

Os registros auxiliares podem funcionar como contadores na função de controle de "loop", onde são decrementados a cada teste de seus conteúdos. Da mesma forma que no endereçamento da memória de dados, somente os nove bits menos significativos são utilizados. A instrução que testa o conteúdo dos registros é a BANZ “Branch on

Auxiliary Register not Zero": se o conteúdo de um dos registros é diferente de zero, então o conteúdo auxiliar testado é decrementado e é realizado o desvio.

Somente os nove bits menos significativos são afetados quando os registros auxiliares são incrementados ou decrementados, numa função de endereçamento ou de controle de "loop". Os bits mais significativos, ou seja, os bits de 9 a 15 nunca são alterados nesta situação, significando que os registros auxiliares funcionam como uma fila circular, isto é, quando alcança seu valor máximo e é realizado um incremento, ele retorna à condição mínima (0). Isto também acontece na situação inversa, ou seja, quando seu valor é zero e é realizado um decremento, o registro fica carregado com a condição máxima (111111111).

Constituído de três bits do registro de status, o apontador de registro auxiliar, ARP, indica qual dos oito registros auxiliares está ativo. Quando o ARP é igual a zero, o corrente registro auxiliar é o AR0, quando ARP = 1, o corrente registro auxiliar é o AR1, e assim, sucessivamente.

3.1.2.4. Registro de status

Consistindo de cinco "flags", o registro de status permite as seguintes informações:

OV "Accumulator Overflow Flag Register": se ocorreu um "overflow" no acumulador, o estado lógico desse "flag" passa a ser igual a 1. Pode-se reinicializar esse "flag" e desviar o contador de programas para o endereço de uma rotina de tratamento de "overflow" através da instrução BV "Branch on Overflow".

OVM "Overflow Mode Bit": o modo "overflow" está desabilitado quando esse "flag" é igual a 0 (zero) e habilitado quando for igual a 1 (um). A instrução ROVM "Reset OVM" carrega o flag OVM com 0 (zero) e a instrução SOVM "Set OVM" carrega o OVM com 1 (um).

INTM "Interrupt Mask Bit": o processador está habilitado a tratar uma interrupção quando o flag INTM possui o valor 0 (zero). O flag indicando o valor 1 (um) informa que o processador está desabilitado. A instrução DINT "Disable Interrupt" carrega o INTM com 1 (um) e a instrução EINT "Enable Interrupt" carrega o INTM com 0 (zero). O flag INTM é automaticamente carregado com o valor 1 (um) antes mesmo de iniciar o serviço da interrupção quando uma solicitação de interrupção é realizada. As instruções EINT e DINT são as únicas que alteram o flag INTM.

ARP "Auxiliar Register Pointer": o valor 0 indica que o ARP está apontando para o registro AR0 e o valor 1 indica que o ARP está apontando para o registro AR1, e assim sucessivamente. As instruções MAR "Modify Auxiliary Register", LARP "Load ARP Immediate", e também as instruções que permitem o modo indireto de endereçamento, permitem que o ARP seja modificado.

DP "Data Memory Page Pointer": a primeira página da memória de dados está ativa, isto é, a página 0 (zero), quando este "flag" está no estado lógico 0 (zero). As

instruções LDP “Load DP” ou LDPK “Load DP Immediate” permitem a modificação do flag DP.

A instrução SST “Save Status Register” possibilita que o registro de status do TMS320C25 seja salvo na memória de dados. Essa instrução, possuindo modo de endereçamento direto, armazena o registro de status na segunda página (página 1) da memória de dados, através do endereço especificado. Para carregar o registro de status pode ser usado a instrução LST. O valor do "flag" de interrupção (INTM) não é modificado por essa instrução.

3.1.2.5. Memória de dados

O TMS320C25 pode endereçar até 64 k palavras de memória de dados externa ao chip. As primeiras 544 palavras de 16 bits, são do tipo "Random Acess Memory" (RAM), interna ao "chip". Os dados não imediatos são armazenados nesta memória. Existem dois blocos que juntos formam 544 palavras, um dos quais podendo ser configurado como memória de dados ou de programa, como mostrado na Figura 3.3.

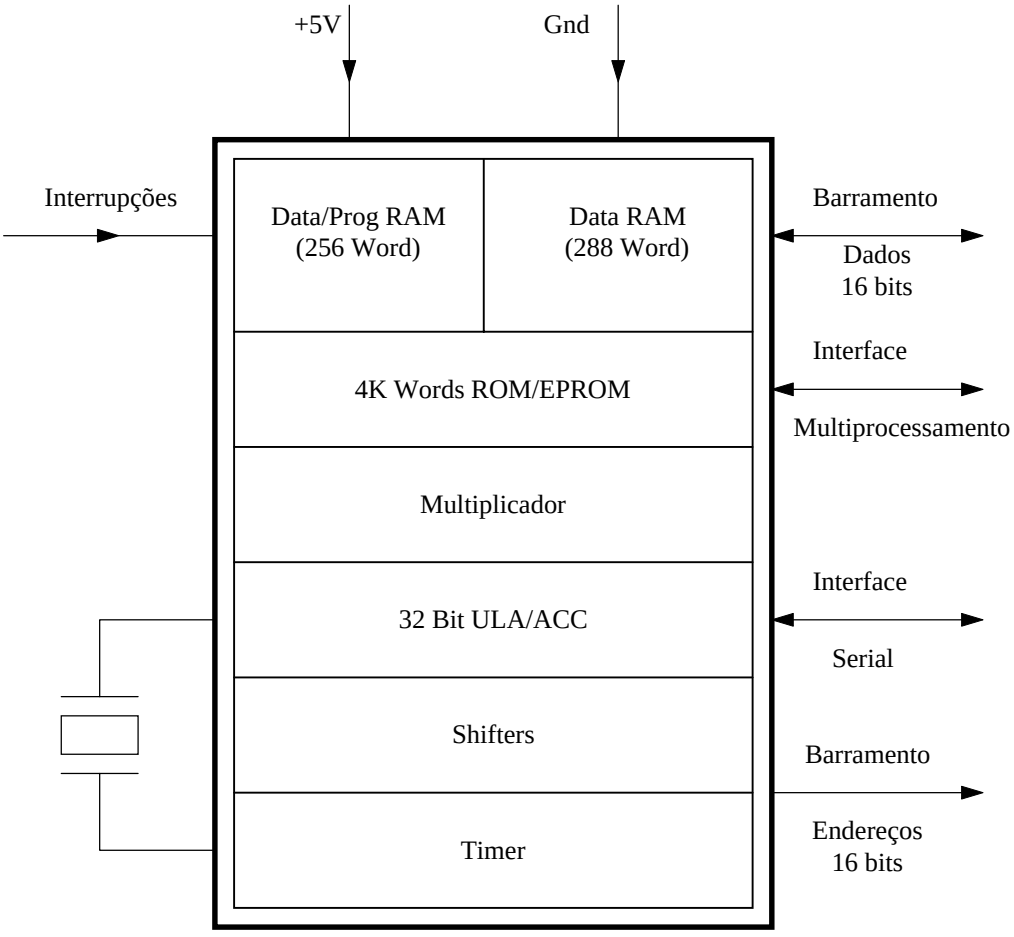


Figura 3.3. Diagrama em blocos simplificado do TMS320C25.

Em alguns momentos, é conveniente armazenar constantes numa memória externa (memória de programa, por exemplo) e carregá-las posteriormente para a

memória de dados. Existem basicamente duas formas de se efetuar a troca de dados com o meio externo: através dos periféricos ou do espaço da memória de programa. As instruções TBLW “Table Write” e TBLR “Table Read” são usadas na troca de dados com a memória de programa externa ao chip. A instrução TBLW transfere dados da memória de dados para a memória de programa, enquanto a instrução TBLR executa a operação inversa, ou seja, transfere dados da memória de programa para a memória de dados. Para serem executadas, elas gastam três ciclos de máquina.

As instruções IN “Input Data” e OUT “Output Data” possibilitam a troca de dados com periféricos. A instrução IN serve para transferir dados provenientes de um periférico para a memória de dados, enquanto que a instrução OUT serve para transferir dados da memória de dados para um periférico. Para serem executadas, elas necessitam de dois ciclos de máquina.

3.1.2.6. Memória de programa

Utilizando-se a memória de programa, externa ao chip, o TMS320C25 consegue endereçar até 65536 palavras de 16 bits, onde os primeiros 4096 endereços são internos ao chip. Apenas um ciclo de máquina é necessário para o acesso a esta memória externa. O mapa da memória de programa do TMS320C25 é ilustrado pela Figura 3.4.

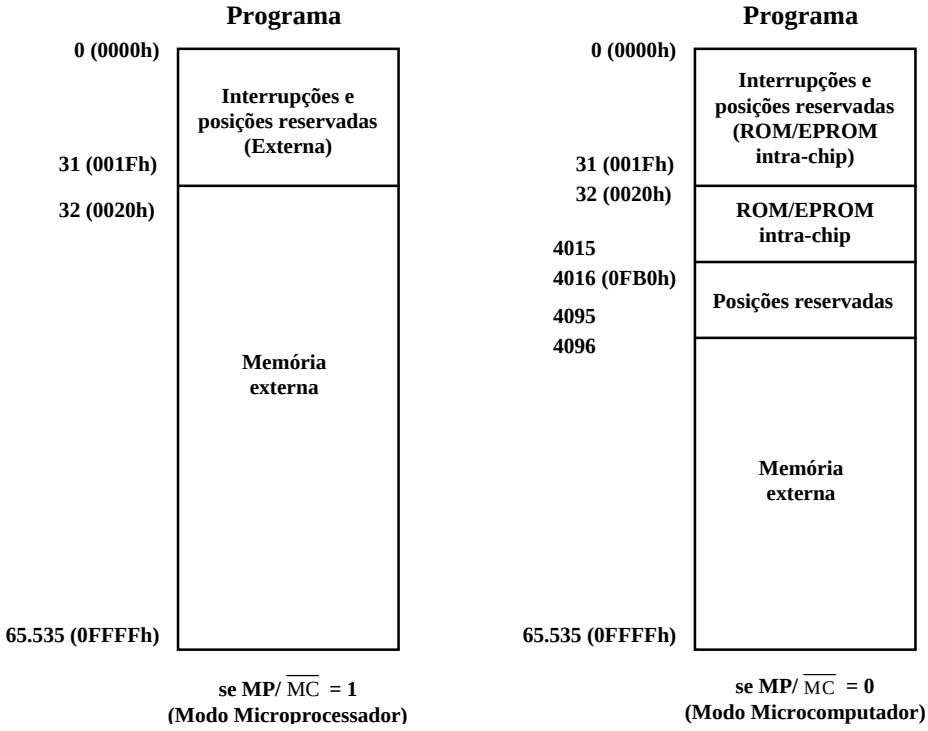


Figura 3.4. Mapa de memória de programa do TMS320C25.

O tratamento de "reset" e interrupção é feito através de uma das trinta e duas primeiras posições reservadas pela Texas para o TMS320C25. Assim, o contador de programas é carregado com o endereço 0000 quando um "reset" ocorre. Com relação a uma solicitação de interrupção, o mesmo procedimento acontece, ou seja, o contador de programas passa a ser carregado com o endereço 0002. Para possibilitar o uso de

instruções de desvios ou "branches", usa-se o intervalo de duas posições de memória entre o endereço de tratamento de "reset" e o endereço do tratamento da interrupção.

3.1.2.7. Contador de programas e pilha

O endereço na memória de programa da próxima instrução a ser executada está contido em um registro de 16 bits denominado contador de programas (PC). Todas as vezes em que a linha de "reset" ($\overline{RS}$) é ativada, o contador de programas é inicializado com 0 (zero), podendo ter ainda seu conteúdo modificado por uma instrução de desvio, quando uma condição for satisfeita.

A pilha do TMS320C25 possui registros de 16 bits com até oito níveis de profundidade, usados para armazenar endereços de retorno de sub-rotinas e interrupções. Se as instruções TBLR ou TBLW não forem executadas, o TMS320C25 pode aninhar até oito sub-rotinas. Se um "overflow" ocorrer na pilha, o nível mais antigo dela será perdido.

Os 16 bits menos significativos do acumulador são armazenados no topo da pilha (TOS) pela instrução PUSH “Push Accumulator On to Stack”. A instrução POP “Pop Top to Accumulator”, por sua vez, faz o trabalho inverso, ou seja, move o topo da pilha para o acumulador.

3.1.2.8. Portas de entrada e saída

Para a comunicação com o meio externo, o TMS320C25 oferece o recurso das portas de entrada e saída. As instruções IN 'Input Data” e OUT “Output Data”, são utilizadas para acessar as portas de I/O do processador. Para fornecerem o endereço das portas de entrada e saída, as quatro primeiras linhas do barramento de endereço (A0 a A3) são multiplexadas, e o restante das linhas de endereço (A4 a A15) permanecem no estado lógico 0 (zero). Desta forma, é possivel endereçar até desesseis portas de entrada ou desesseis de saída.

Para indicar que o TMS320C25 está pronto para receber um dado oriundo de um periférico, é gerado o sinal “$\overline{R}$” “Read Enable” na execução de uma instrução IN. Esse sinal só é ativado na execução dessa instrução.

Também é gerado um sinal para indicar que o processador vai transferir um dado da memória de dados para um periférico de saída, na execução de uma instrução OUT. Esse sinal é o “$\overline{W}$” “Write Enable”. Ele é ativado no segundo ciclo dessa instrução. A instrução TBLW “Table Write” é outra instrução que ativa o sinal “$\overline{W}$”. A diferença entre o uso das instruções TBLW e OUT está no momento em que o sinal “$\overline{W}$” é ativado, ou seja, quando uma instrução TBLW é executada o sinal “$\overline{W}$” é ativado no terceiro ciclo da instrução. Quando o sinal “$\overline{W}$” é ativado, o barramento de dados do TMS320C25 sai do estado de alta impedância.

3.1.2.9. Interrupção

Para garantir a suspensão de um processo para atender outro de alta prioridade, lança-se mão de um recurso denominado interrupção. Para se requerer uma interrupção, deve-se aplicar um nível lógico 0 (zero) no pino INT por, no mínimo, um ciclo de "clock". O valor corrente do contador de programa é colocado no topo da pilha, o endereço 0002 da memória de programa é carregado no contador de programas e o "flag" INTM tem seu estado lógico alterado para o valor 1 (um), imediatamente após o reconhecimento de uma interrupção pelo processador.

Através da instrução DINT “Disable Interrupt”, a interrupção pode ser mascarada em pontos críticos do programa, enquanto que a instrução EINT “Enable Interrupt” permite que a interrupção possa ser desmascarada.

3.1.2.10. Pino BIO

Para a monitoração de periféricos o TMS320C25 dispõe de um recurso denominado pino BIO. A execução da instrução BIOZ “Branch I/O Status Equal to Zero” causa um desvio para o endereço especificado na instrução quando um nível lógico 0 (zero) está presente neste pino. Nos processos com tempo crítico, esse recurso pode ser uma alternativa ao uso de interrupção. A cada ciclo da máquina o pino BIO do TMS320C25 é amostrado.

3.1.3. A eletrônica do processador digital de sinais TMS320C25

Nesta seção o Processador Digital de Sinais TMS320C25 terá todos os sinais eletrônicos identificados, e suas funções definidas, permitindo, desta forma, um melhor entendimento do seu hardware.

3.1.3.1. Sinais A0/PA0-A11

O TMS320C25 pode endereçar até 4096 posições de memória de programa, de 16 bits cada, quando os sinais A0-A11 são usados. As quatro primeiras linhas de endereço A0-A3 também contêm o endereço (PA0-PA3) de uma das portas de entrada e saída do TMS320C25, durante a execução das instruções IN “Input Data” e OUT “Output Data”. O restante das linhas de endereço A4-A11 assumem o nível lógico 0 (zero), nesse último caso. As linhas de endereço A0-A11 nunca vão para o estado de alta impedância, ou seja, estão sempre ativas.

3.1.3.2. Sinais de controle

Os sinais “$\overline{R}$” “Read Enable” e “$\overline{W}$” “Write Enable”, que são mutuamente exclusivos, isto é, nunca estão ativos no mesmo intervalo de tempo, constituem os dois sinais de controle do TMS320C25.

Para realizar uma transferência de dados da memória de dados para um periférico de saída, ou para a memória de programa, o TMS320C25 deve ser habilitado pelo sinal "$\overline{W}$". Somente quando uma instrução OUT "Output Data" ou TBLW "Table Write" é executada o sinal "$\overline{W}$" é ativado ($\overline{W}$ = 0). Outra característica do sinal "$\overline{W}$" é que ele é ativado no primeiro ciclo da instrução OUT e no segundo ciclo da instrução TBLW.

Somente no segundo ciclo da instrução IN "Input Data" o sinal "$\overline{R}$" ($\overline{R}$ = 0) é ativado, e o TMS320C25 está habilitado para receber dados oriundos de um periférico de entrada quando o sinal "$\overline{R}$" está ativo ($\overline{R}$ = 0).

3.1.3.3. Clock

Visando uma maior flexibilidade, o "clock" do TMS320C25 pode ser gerado tanto pelo seu oscilador interno quanto por um oscilador externo.

3.1.3.4. D0-D15

As linhas do "Bidirectional Data Bus" do TMS320C25, denominadas de D0-D15, estão sempre no estado de alta impedância, exceto quando o sinal de "$\overline{W}$" está ativo ($\overline{W}$ = 0).

3.1.3.5. RS "Reset"

Para reinicializar um processamento ou colocar o TMS320C25 em estado de "reset" indefinidamente, usa-se a linha de entrada "$\overline{RS}$". Quando é fornecido à entrada "$\overline{RS}$" um sinal de nível lógico 0 (zero), de duração mínima de cinco ciclos de "clock", os sinais de controle "$\overline{W}$" e "$\overline{R}$" são forçados a assumirem o nível lógico 1 (um), e as linhas de endereço A0/PA0-A11 e o contador de programas (PC) assumem o estado lógico 0 (zero). Um "reset" no TMS320C25 também desabilita interrupções e limpa o "flag" associado a elas.

3.1.3.6. Interrupção

Quando um sinal de nível lógico 0 (zero) é fornecido às entradas INT0, INT1 ou INT2, é gerada uma interrupção no TMS320C25. Tanto um periférico como o microcomputador podem fornecer este sinal, quando o TMS320C25 está acoplado ao seu barramento.

3.1.3.7. BIO "I/O Branch Control"

Na monitoração de periféricos, o "I/O Branch Control" é outro recurso disponível no TMS320C25. A instrução BIOZ "Branch If BIO = 0" testa o estado do sinal de entrada "$\overline{BIO}$", e se este equivale ao nível lógico 0 (zero), um desvio é efetuado no programa, para o endereço fornecido na instrução. Igualmente às entradas INT0, INT1 ou INT2, o sinal da entrada "$\overline{BIO}$" pode ser fornecido por um periférico ou pelo microcomputador.

3.1.4. Novos processadores de sinais

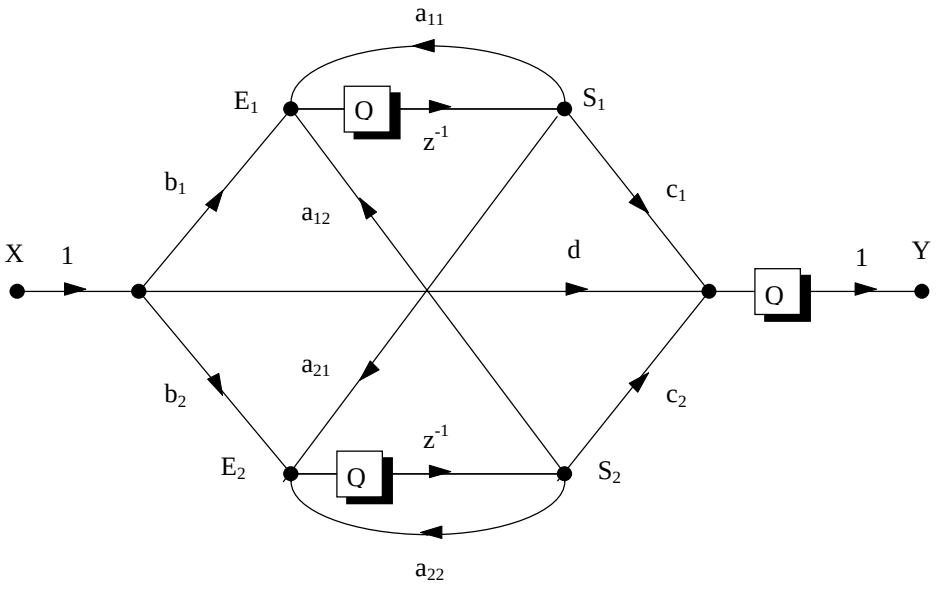
A Texas Instruments também vem fabricando outros membros da família TMS, muito mais velozes, que além de possuírem mais memória e um conjunto maior de instruções podem operar também com aritmética de ponto flutuante, como é o caso das famílias TMS320C3X e TMS320C4X, este último incluindo fatores específicos para a implementação de processamento paralelo. Também já está disponível o TMS320C50, uma nova família de processadores de ponto fixo, mais veloz e com acesso à memória mais versátil.

Processadores de outros fabricantes também têm disputado este segmento de mercado, tais como o DSP56000, da Motorola, o ADSP2100, da Analog Devices, e o μPD77230 da NEC /Mor86/, todos com ciclo de instrução de 100 ns. O DSP56000, por exemplo, trabalha com comprimento de palavra de 24 bits e o μPD77230 tem aritmética de ponto flutuante de 24+6 bits. Ambos permitem uma faixa dinâmica muito maior, o que é muito importante em aplicações críticas de áudio.

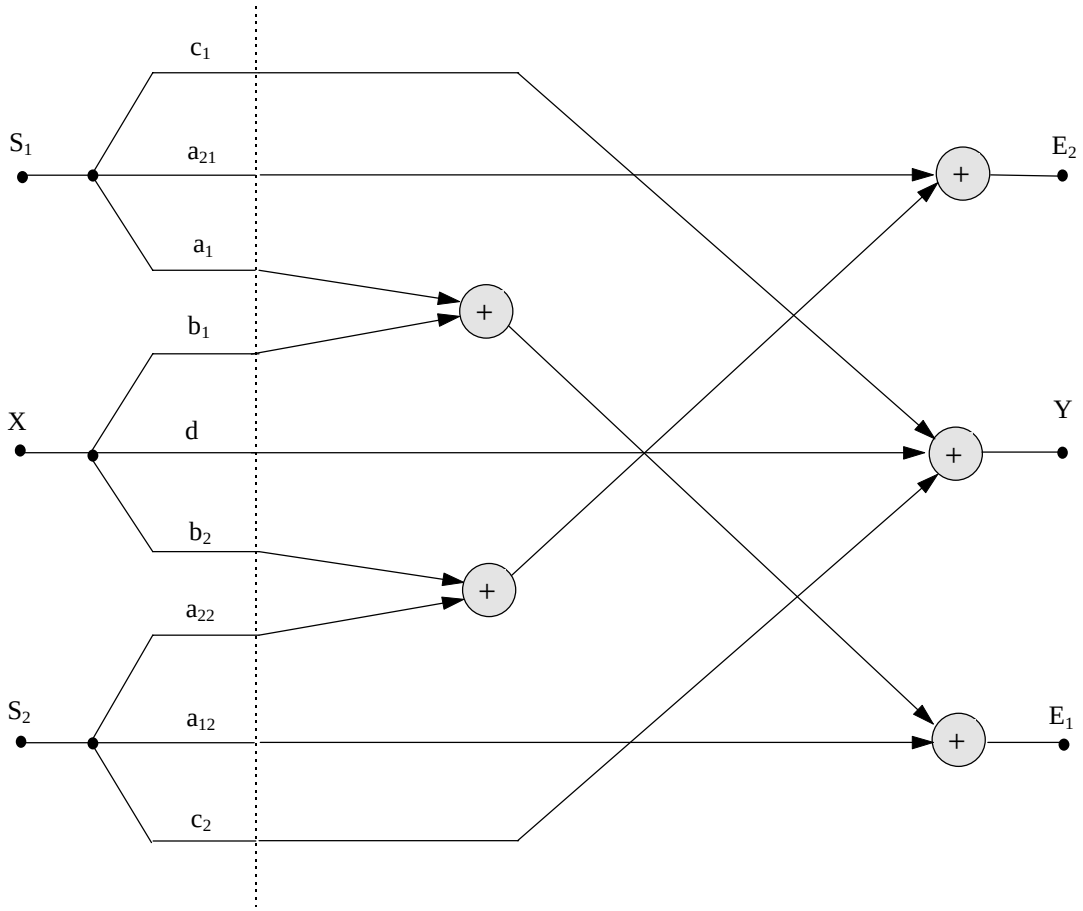
3.2. Implementação de filtros digitais usando o processador TMS320C25

O problema de se encontrar o programa mais eficiente (mais rápido) para implementar um algoritmo de filtragem digital, dado o hardware adotado, é uma tarefa bastante árdua. Crochiere e Oppenheim dão um método para determinar as relações precedentes para um dado algoritmo ou grafo de um filtro /Cro75/. Essas relações dão a ordem (ou ordens) na qual as operações podem ser executadas.

Quando um filtro é implementado através de blocos modulares de segunda ordem, é usualmente fácil escrever um código bastante eficiente. Consideremos a codificação de uma seção de filtro digital realizada sob a forma de estrutura de mínimo ruído. O grafo de fluxo da seção é mostrado na Figura 3.5.a.



a) Grafo de fluxo



b) Grafo de computação

Figura 3.5 a) Grafo de fluxo e b) Grafo de computação para a estrutura de rede ótima.

Durante cada intervalo de amostragem, lê-se uma nova entrada, computa-se a nova saída e atualiza-se os atrasos. Do ponto de vista da programação, existem 3 entradas: a entrada global e os valores das duas variáveis de estado (saídas dos atrasos). Sejam elas X, S_1 e S_2, como mostrado na Figura 3.5.a. Existem, portanto, 3 saídas: a saída global Y e os novos valores das saídas dos atrasos, E_1 e E_2. As saídas são computadas a partir das entradas, com sucessivas multiplicações e adições.

Colocando as entradas do lado esquerdo e as saídas do lado direito, pode-se desenhar o grafo de computação da Figura 3.5.b, que mostra a ordem na qual as operações podem ser executadas. As operações que podem ser executadas em paralelo são verticalmente alinhadas. É fácil verificar, inclusive, como uma implementação pode ser obtida com o máximo de paralelismo. O grafo de computação ilustrado pela Figura 3.5.b é usado para definir o algoritmo, que se traduzirá em um programa "assembly" próprio do DSP, definindo as inúmeras operações de multiplicação e adição envolvidos na implementação de uma referida estrutura de filtro digital.

Um bom tutorial para programação com processadores TMS320C25 pode ser encontrado em /Tex93/. A seguir, alguns problemas especiais com o TMS320C25 são considerados. Ao final do capítulo, a programação de estruturas de segunda ordem é também discutida.

3.2.1. Implementando a multiplicação

Para examinar a multiplicação em detalhes, usando o TMS320C25, seja a multiplicação de um valor de sinal X por um coeficiente do filtro A. Sejam ambos menores que 1 em magnitude, com representação binária em complemento a dois em 16 bits, havendo um bit de sinal e 15 bits para a parte fracionária de cada valor. Então, uma única instrução realizará a multiplicação. Se o coeficiente for maior que 1 em magnitude, a parte inteira será implementada com adições (ou subtrações), de forma que a multiplicação será mais elaborada, e consequentemente mais lenta. Isso será visto adiante.

Efetivamente, os coeficientes são escritos no programa como valores inteiros na faixa $[-2^{15}..2^{15}-1]$ ou [-32768..32767] (se coeficientes de 13 bits para multiplicação imediata são usados, eles são quantizados na faixa [-4096..4095]). A representação às vezes é chamada de Q15 e Q12, respectivamente. O valor decimal exato A_E e o coeficiente quantizado A são relacionados do seguinte modo:

$$A = round[(A_E \bmod 1).32768] \tag{3.1.a}$$

$$A_E \sim integ[A_E] + \frac{A}{32768} \tag{3.1.b}$$

onde "round" denota o arredondamento, "A mod B" denota o resto da divisão de A por B, e "integ" denota a parte inteira do argumento. Tomemos como exemplo $A_E = 1.77$. Assim, temos

$$A = round[(1.77 \bmod 1).32768] = round[0.77.32768] = 25231 \tag{3.1.c}$$

$$A_E \sim 1 + \frac{25231}{32768} = 1.769989 \tag{3.1.d}$$

Ambos, coeficiente e valor do sinal, estão na memória de dados, e deseja-se que o resultado seja armazenado na variável Y, também na memória de dados. A multiplicação é feita como se segue:

```
        LT      X       Carrega X no registro T
        MPY     A       Multiplica X por A
        PAC             O produto é acumulado
        ADD     X, 15   Parte inteira, repete se for
                        necessário (SUB, se A < 0)
        SACH    Y, 1    Armazena em Y
```

Quando do armazenamento em Y, um deslocamento para a esquerda é necessário, para eliminar o bit extra de sinal, que vem com a multiplicação /Laa87/.

3.2.2. Implementando o truncamento em magnitude

Quando o produto é armazenado, depois da multiplicação, ele é quantizado para um comprimento de palavra menor, de 16 bits. Se os bits menos significativos forem simplesmente descartados, o resultado da quantização é um truncamento em complemento a dois.

Outras regras de quantização podem ser implementadas pela adição de instruções extra. O arredondamento, por exemplo, é obtido quando a metade do passo da quantização é apropriadamente adicionado ao acumulador, antes do armazenamento. Desta forma, armazenando na variável `ONE` da memória de dados um bit menos significativo (LSB), tem-se:

```
        LT          X          Carrega X no registro T
        ..
        ..
        ..
        ADD         ONE, 15    Soma Q/2, para efetuar o
                               arredondamento
                               (ONE = 0000 0000 0000 0001)
        SACH        Y
```

A implementação do truncamento em magnitude é um pouco mais complicada, pois requer que a magnitude do sinal seja reduzida. Com valores positivos de sinal, o truncamento em magnitude fica igual ao truncamento em complemento a dois (descarta-se os bits menos significativos), mas com valores negativos é necessário um passo a mais, nesse tipo de quantização. Ele deve ocorrer antes da armazenagem, e corresponde a uma instrução de desvio, como a seguir

```
        LT          X     Carrega X no registro T
        ..
        ..
        ..
        BGEZ        POS   Desvia se acc >= 0
        ADDH        ONE   Adiciona Q, para truncamento em
                          magnitude
POS     SACH        Y,1   Armazena em Y
```

onde POS indica um endereço na memória de programa. Note-se que esta implementação, ao invés de requerer 2 ciclos extras de instruções (como no caso do arredondamento), passa a requerer 3 ciclos, pois a instrução de desvio, BGEZ, necessita de 2 ciclos.

3.2.3. Um exemplo de programa

Como forma de ilustrar a programação de um filtro digital no DSP TMS320C25, o Apêndice A traz o programa “assembly” correspondente a uma estrutura paralela de até cinco blocos como os da Figura 2.7 (estrutura ótima), onde a programação de cada bloco segue o grafo de computação mostrado na Figura 3.5.

Alguns pontos devem ser destacados em tal programa: o primeiro é a implementação das multiplicações, prevendo que o módulo dos coeficientes multiplicadores do vetor **c** podem ser maiores que 1, e do truncamento em magnitude, conforme discutido acima. Observe-se que a última soma, na geração dos valores atualizados dos estados, usa o recurso da saturação aritmética. O segundo ponto de destaque é que o programa não se refere a um filtro em particular. Ao contrário, ele se refere a uma família inteira, cujos coeficientes, número de blocos e freqüência de amostragem particulares são repassados para uma tabela na memória de dados do DSP através de uma interface adequada /Sar94/, incluída na **Plataforma Integrada para Processamento Digital de Sinais - PIPDS**. A vantagem de tal interface é que um único programa fonte é escrito, e um único programa executável é montado, para uma família inteira de filtros. Observe-se, finalmente, que os coeficientes do filtro, carregados via tal interface, já são devidamente formatados para 16 bits, usando faixa dinâmica de valor um e arredondamento.

Um outro ponto a ser destacado, no programa que implementa o filtro digital no DSP TMS320C25, é a utilização de dois "buffers" na memória de dados, de comprimento 2048 palavras cada um, onde as amostras de x[n] e y[n] são armazenadas, em forma de fila circular, para serem lidos e processados pela plataforma PIPDS, da forma descrita no Capítulo 4.

3.3. Critérios usados para avaliar o desempenho de uma estrutura de filtro digital

No projeto de filtros digitais existem três passos básicos a serem seguidos, nominalmente /Ren82/

1. o projeto da função de transferência ;
2. o projeto da estrutura de rede adotada para o algoritmo e
3. a definição do hardware adotado.

Os três passos não são independentes. O "hardware" existente pode restringir o tipo de filtro ou sua ordem, ou a aplicação pode necessitar de uma freqüência de amostragem muito elevada que requeira um "hardware" especial, ou ambos. É desejável, de qualquer maneira, o projeto de filtros que apresentem um bom desempenho com o mínimo custo.

Uma outra diferença importante está no custo de uma multiplicação. Enquanto na implementação com "hardware" de uso genérico (microcomputadores) a multiplicação toma muito mais tempo que uma adição, nos processadores de sinais de uso dedicado (DSP's) esta operação gasta o mesmo tempo. Existem algoritmos que minimizam o número de multiplicações e adições podendo maximizar, assim, o desempenho dos processadores de sinais /Asp84/. De qualquer forma, o uso do DSP já melhora o desempenho, sob o aspecto da velocidade de processamento, já que tal velocidade (ou a máxima taxa de amostragem obtida para um filtro) é inversamente proporcional ao número de instruções a serem executadas para cada amostra da entrada. A questão de como tornar a estrutura do filtro mais eficiente, então, faz com que o código "assembly" do filtro também desempenhe um papel crucial.

Usando os processadores de sinais, os critérios mais importantes a serem considerados na implementação dos filtros são

1. a máxima freqüência de amostragem ;
2. a memória de dados necessária e
3. a memória de programa necessária.

No caso presente, a placa utilizada permite amostragem em até 200 kHz, com até 64k palavras de memória de dados e 64k palavras de memória de programa /Dal90/.

Quando a função de transferência do filtro projetado atende às especificações, é ainda necessário um bom desempenho quando da sua implementação com comprimento de palavra finito. Usualmente o bom desempenho do filtro recursivo em tal situação pode ser caracterizado por

1. baixa sensibilidade à variação dos coeficientes ;
2. baixo ruído na saída devido à quantização do produto e
3. ausência garantida de oscilações periódicas (ciclos limite) em qualquer situação.

Para tanto, a solução é a seleção adequada das estruturas a serem usadas, como discutido no Capítulo 2.

A modularidade também se constitui numa característica importante da estrutura de um filtro, quando implementada com um processador de sinais. Se a estrutura puder ser formada por subestruturas que podem ser carregadas separadamente (tais como os blocos de segunda ordem de redes paralelas), a tarefa de programação será muito facilitada, e menos exposta a erros.

No presente trabalho, todas as questões relacionadas ao bom desempenho do filtro digital implementado em DSP são consideradas adequadamente. Isso é garantido pela seleção adequada das estruturas (as quatro estruturas detalhadas no Capítulo 2 são as que serão utilizadas, pois os filtros projetados serão de banda estreita) e pela sua programação correta, buscando minimizar o tempo dispendido no cálculo de cada amostra de saída. Inclusive, como forma de comparar o tempo necessário para computar cada uma das estruturas programadas, e assim ter uma idéia da sua complexidade computacional, uma tarefa de medição de tempo foi incorporada à plataforma PIPDS desenvolvida (ver Capítulo 4), usando o temporizador interno ao DSP /Tex93/. Observe-se, também, que tal tarefa pode ser usada como ferramenta auxiliar no desenvolvimento adequado dos programas “assembly” correspondentes a cada rede implementada.

CAPÍTULO 4

Desenvolvimento da Interface Gráfica de Usuário

Este capítulo descreve a **Plataforma Integrada para Processamento Digital de Sinais - PIPDS**, em termos de sua concepção e sua implementação como um ambiente integrado para o desenvolvimento de sistemas de processamento digital de sinais. Tal descrição tem por base o fluxograma ilustrado na Figura 1.3, que mostra as diversas etapas do projeto de filtros digitais.

4.1. Os requisitos de um ambiente para projeto e implementação de filtros digitais

A plataforma deve ser construída de tal forma que auxilie o projetista na tarefa de projetar e implementar um filtro digital em um hardware dedicado, à base de DSP. Para alcançar este objetivo necessita-se, portanto, de um ambiente com características que atendam às condições abaixo relacionadas:

- facilidade de uso, baseada preferencialmente em interface gráfica;
- capacidade de execução das operações fundamentais ao desenvolvimento da aplicação;
- baixo custo de implementação;
- total integração, permitindo que o projetista, além de projetar o filtro dentro do ambiente de um IBM-PC, possa verificar a correção do programa rodando no processador dedicado.

Quando comparada com outros programas de projeto de filtros disponíveis comercialmente /Tex93, Cha90/, esta plataforma se destaca principalmente por incluir a parte de implementação dos filtros em DSP, além de se constituir em um ambiente de múltiplas tarefas, dentro das possibilidades do ambiente Windows /Pet93, Yao95, Swa94/, para o qual é desenvolvida.

As principais tarefas a serem desempenhadas dentro da interface desenvolvida são descritas nas sub-seções a seguir.

4.1.1. Especificação, aproximação e síntese do filtro digital

Deve ser escolhido o tipo de filtro que se quer projetar, dentre passa-baixas, passa-altas, passa-faixa e rejeita-faixa. A partir dessa especificação, são definidos os parâmetros característicos do filtro (gabarito de atenuação ou formatação do espectro). Uma vez aproximado o filtro, os coeficientes de sua função de transferência H(z) deverão ser salvos em arquivo.

O filtro, posteriormente, deve ser sintetizado. Para isso, é selecionado um entre vários tipos de estrutura disponível para implementar o algoritmo. O projetista, então, deve decidir qual o tipo de estrutura e qual a forma mais conveniente (cascata ou paralela) para um dado filtro. Deve ser disponibilizado um grupo mínimo de estruturas de segunda ordem, de desempenho satisfatório, para conexão cascata ou paralela (ver

Capítulo 2), mas a capacidade de inserção de novas estruturas desejadas pelo projetista deve ser garantida.

Também é necessário fazer a quantização dos coeficientes do filtro, para uma representação compatível com o tamanho dos registros do DSP. Para isso, deve ser informado o arquivo contendo a rede anteriormente sintetizada, assim como o número de bits usado.

Como se vê, as operações até aqui desenvolvidas usam somente a CPU do PC, e demandam do usuário algum tipo de seleção de opções, algum dado numérico adicional e nomes de arquivos para leitura ou escrita de informações. Tais opções, então, devem ser oferecidas ao usuário via alguma interface amigável, no espaço de trabalho correspondente ao computador PC.

4.1.2. Ferramentas de avaliação do filtro projetado

A plataforma deve oferecer, também, ao projetista, meios para que ele possa analisar graficamente o comportamento do filtro implementado em hardware dedicado, ou mesmo das estruturas sintetizadas. Desta forma, gráficos da resposta em freqüência do filtro devem ser gerados. Estes gráficos são construídos a partir de informações fornecidas pelo projetista, tais como as estruturas das redes e os arquivos que as contêm.

Os gráficos básicos são gerados a partir da função de transferência que o projetista pretende implementar, assim como da rede sintetizada, ambas armazenadas anteriormente em arquivos.

Outra opção é plotar o gráfico de resposta ao impulso do filtro projetado: para isso, após programar o filtro no DSP, o projetista deverá poder dispor da FFT das amostras colhidas a partir da aplicação, na entrada do filtro, de uma função impulso com a amplitude desejada.

A utilização de tais ferramentas permitirá ao projetista verificar a adequação do filtro projetado ao gabarito de atenuação desejado, via o gráfico da função de transferência; verificar a correção do programa escrito para o DSP, via o gráfico da resposta em freqüência obtido a partir da resposta ao impulso, ou observar os efeitos da realização em precisão finita, tanto via o gráfico da resposta em freqüência da rede sintetizada como via o gráfico da resposta em freqüência obtido a partir da resposta ao impulso. Por fim, é ainda conveniente poder visualizar em detalhes uma determinada região de interesse de tais gráficos, através de uma ferramenta de zoom. Novamente, todo o processamento fica restrito à CPU do PC, com excessão do caso de se desejar tratar a resposta ao impulso do filtro. Nesse caso, é necessário obter as amostras de tal resposta, o que exigirá uma implementação particular do programa do filtro no DSP, já que não há necessidade de sua execução sincronizada com um sinal externo. Os dados assim obtidos devem ficar acessíveis à CPU do PC, que gerará e exibirá o gráfico desejado. Novamente, a interface com o usuário é feita pelo PC, que agora também necessitará interagir com o DSP.

4.1.3. Execução do filtro em tempo real

O objetivo final do projeto, na abordagem aqui discutida, é ter um programa rodando no DSP, para realizar o filtro sintetizado em tempo real. Em tal situação, o projetista normalmente não tem como verificar se o algoritmo em execução está correto, pois não há, em geral, nenhuma interface com o processo em tempo real. Nesse aspecto, seria conveniente um sistema gráfico que permitisse ao projetista visualizar o filtro desenvolvido, tanto no tempo (tipo um osciloscópio) quanto em frequêcia (tipo um analisador de espectros). Novamente, o PC faria não só a interface com o usuário, mas também com o DSP, para buscar a informação desejada pelo projetista.

Observe-se, aqui, que há a disponibilidade de dois processadores: o DSP e a CPU do PC hospedeiro. Porém eles devem atuar de forma completamente independente: o DSP ficará responsável exclusivamente pela execução, sincronizada pelo período de amostragem, do algoritmo correspondente ao filtro implementado, enquanto o processador hospedeiro faz toda a interface com o usuário, inclusive gráfica.

4.1.4. Compilação/Depuração do programa escrito para o DSP

Considerando que o filtro projetado será traduzido em um programa, será necessária a sua compilação e a transferência do código gerado para a memória de programa do TMS320C25. Adicionalmente, é importante a depuração do referido programa, conforme sugere o fluxograma da Figura 1.3, como ferramenta de apoio ao desenvolvimento do filtro. Destaque-se que a única linguagem disponível com a placa de DSP utilizada é a linguagem assembly, o que leva à geração de arquivos ".asm" para os programas. Assim, as ferramentas de depuração adequadas são um "disassembly", a visualização dos registros do DSP e de sua memória de dados, execução com "breakpoints" (ou passo a passo do programa), etc /Dal90/. Observe-se, aqui, o papel do PC como elemento de interface com o usuário e com o DSP, assim como o fato de que, nesse caso, o DSP não pode funcionar em tempo real, pois seria perdido o sincronismo entre as amostras.

4.1.5. Transferência dos coeficientes de um dado filtro para a placa de DSP

O projetista, depois que projeta um filtro digital, deseja implementar o referido filtro no processador dedicado disponível na placa conectada ao barramento do PC. Para isso, a melhor opção é programar uma família inteira (ou uma classe) de filtros, através de um programa adequadamente escrito. Isto feito, o projetista informaria qual a freqüência de amostragem para o filtro particular implementado, para que se possa programar os conversores A/D e D/A, além do tipo de configuração dos blocos de segunda ordem, assim como os coeficientes de tais blocos, que seriam carregados na memória de dados do DSP. Só então o programa seria ativado. Isso foi, inclusive, uma das motivações para o presente trabalho /Sar94/.

4.1.6. Edição de arquivos ASCII

Durante o desenvolvimento dos programas para os filtros, o projetista certamente necessitará manipular arquivos ASCII (ver fluxograma da Figura 1.3). É prevista a capacidade de manipular, inclusive imprimir, arquivos contendo programas fonte para o

DSP, assim como arquivos contendo coeficientes de funções de transferência e/ou coeficientes das redes implementadas.

Aqui, novamente, o processamento é restrito à CPU do PC, e uma interface amigável com o usuário é essencial.

4.2. O uso de interfaces gráficas de usuário

Dada a formulação básica do problema, decidiu-se desenvolver uma Interface Gráfica de Usuário (GUI), como forma de dotar o projetista de filtros digitais das ferramentas gráficas mencionadas. Existem várias razões que levaram a adotar uma interface gráfica de usuário como ambiente de desenvolvimento para a plataforma PIPDS.

Algumas vezes também chamada de “interface visual” ou ambiente gráfico baseado em janelas, a interface gráfica de usuário tem seus conceitos datados da década de 70, com o trabalho pioneiro desenvolvido no Centro de Pesquisas da Xerox (PARC), em Palo Alto, para máquinas como o Alto e o Star e ambientes como o Smalltalk. Ela é uma das mais revolucionárias mudanças que ocorreram na evolução dos sistemas computacionais. Essa revolução tem aumentado a acessibilidade e o uso de computadores pelo público em geral. Exemplo disso é a inteface gráfica baseada no familiar Windows, com seus ícones, menus, dispostivos apontadores, etc.

Umas das características mais comuns encontradas em quase todas as interfaces gráficas do usuário é o fato de utilizarem gráficos em monitores, formados por mapas de bits. Os gráficos fazem uma melhor utilização da tela, transmitindo informações de uma maneira visual mais rica, e posssibilitam o que se chama de WYSIWYG (“what you see is what you get” - o que você vê é aquilo que você obtém) das figuras e do texto formatado para o documento que será impresso. Isto foi um fator determinante a ser levado em consideração quando da implementação da plataforma.

O monitor de vídeo era utilizado, antigamente, na maioria das vezes, para reproduzir o texto que o usuário digitava no teclado. Em uma interface gráfica do usuário, ele se transforma em uma fonte de entrada de dados do usuário: a tela do vídeo agora mostra vários objetos gráficos na forma de ícones e dispositivos de entrada, como botões e barras de rolagem. Usando o teclado ou, mais diretamente, um dispositivo apontador (como um mouse), o usuário pode, diretamente, manipular esses objetos na tela: barras de rolagem podem ser roladas, objetos gráficos podem ser arrastados e os botões podem ser pressionados. Torna-se, assim, mais íntima a interação entre o usuário e o programa: em vez da via de mão única do teclado para o programa e daí para a tela do vídeo, o usuário passa a interagir diretamente com os objetos na tela.

Inicialmente, no fracassado computador Lisa e um ano mais tarde no bem sucedido Macintosh, lançado em janeiro de 1984, o trabalho desenvovido pela Xerox foi aplicado e popularizado pela Apple Computers. O Apple Macintosh continua sendo um grande competidor do IBM-PC no mercado dos computadores pessoais, o que se deve mais ao seu sistema operacional, que o torna extremamente atrativo aos usuários.

Novas interfaces gráficas do usuário têm sido criadas na indústria dos computadores pessoais e não-pessoais, desde o lançamento do Macintosh. Para os computadores compatíveis com o IBM, usando o MS-DOS, existe o Windows. Para os compatíveis usando o OS/2, existe o Presentation Manager. Para o Amiga da Commodore, existe o Intuition. Para o Atari, existe o GEM. Para máquinas que rodam o sistema operacional UNIX, existe o sistema X-Windows. Para as estações de trabalho Sun Microssystems, existe o NeWS. Para o computador NeXT, existe o NextStep.

Entre as várias interfaces gráficas de usuário existentes foi escolhida a interface baseada no MS-WINDOWS, para o desenvolvimento da plataforma PIPDS, pois o Microsoft Windows vem se destacando como um dos mais conhecidos ambientes de interface gráfica do usuário para o MS-DOS, desde seu lançamento, em novembro de 1985. Milhões de cópias do programa já foram vendidas e centenas de aplicativos Windows já estão disponíveis.

Existem inúmeras vantagens, para o usuário e para o programador, decorrentes da adoção deste tipo de ambiente, entre as quais podemos relacionar:

- o Windows cria, para o usuário, um ambiente gráfico multitarefas com janelas que executam programas feitos especificamente para o Windows;
- mais fáceis de aprender e usar do que os programas convencionais para o MS-DOS, os programas escritos para o Windows têm uma mesma aparência e uma mesma estrutura de comandos;
- os usuários podem facilmente transitar entre os diferentes programas Windows, e transferir dados de um para o outro;
- o Windows também contém um Gerenciador de Programas, para permitir a execução de programas, um Gerenciador de Arquivos, para manutenção nos arquivos e um Gerenciador de Impressão, para o gerenciamento da fila de impressão.

O Windows ainda fornece ao programador recursos tais como:

- inúmeras rotinas internas que permitem o uso de menus, caixas de diálogo, barras de rolagem e outros componentes de uma interface amigável;
- uma extensa linguagem de programação de gráficos que inclui o uso de texto formatado em diversas fontes;
- tratamento do teclado, do mouse, do monitor, da impressora, do relógio e das portas de comunicação RS-232 de uma maneira independente do dispositivo. Em outras palavras, em diferentes configurações de hardware os programas Windows rodam da mesma maneira.

Em novembro de 1983 o Windows foi anunciado pela Microsoft, sendo lançado dois anos mais tarde, em novembro de 1985. Nos dois anos seguintes, o Windows 1.01 (a primeira versão) foi seguido por várias atualizações, para atender ao mercado internacional e para incluir controladores para novos monitores de vídeo e impressoras.

Em novembro de 1987 o Windows 2.0 foi lançado. Para torná-lo mais consistente com o vindouro Presentation Manager do OS/2 (lançado em outubro de 1988), essa versão incorporou diversas alterações na interface com o usuário. A

alteração mais significativa envolvia o uso de janelas que se sobrepunham, em vez de janelas dispostas lado a lado, como nas versões anteriores. O Windows 2.0 também incluiu melhoramentos na interface do teclado e do mouse, particularmente para os menus e caixas de diálogo.

Lançado logo após o Windows 2.0, o Windows/386 usava o modo Virtual-86 do microprocessador 80386 para colocar em janelas e executar simultaneamente vários programas do DOS que acessavam o hardware diretamente. Por simetria, o Windows 2.1 foi renomeado para Windows/286.

Em 22 de maio de 1990 o Windows 3 foi apresentado ao público. As versões anteriores Windows/286 e Windows/386 foram unidas em um único produto, com esta versão. Está no suporte ao modo de operação protegido dos microprocessadores Intel 80286 e 80386 a grande alteração contida no Windows 3: isso permitiu o acesso a 16 Mbytes de memória. Também foram totalmente remodelados os programas de interface (Gerenciador de Programas, Gerenciador de Tarefas e Gerenciador de Arquivos).

O Microsoft Windows 3.1, por sua vez, foi lançado em abril de 1992. O recurso novo mais significativo nesta nova versão é a tecnologia de fontes TrueType, desenvolvida pela Apple Computers e pela Microsoft: ela permite o uso de fontes de contorno escaláveis no Windows. A nova versão também inclui o suporte à Multimídia (som), à Incorporação e Vinculação de Objetos (OLE, do inglês Object Linking and Embedding) e às caixas de diálogo comuns. O Windows 3.1 roda somente no modo protegido e requer pelo menos o microprocessador 80286, com no mínimo 1MB de memória RAM.

O centro da estratégia da Microsoft para sistemas operacionais agora é o Windows. A empresa tem o Windows como alvo para tudo, desde pequenos computadores portáteis que cabem na palma da mão, até poderosas estações de trabalho RISC (do inglês, Reduced Instruction Set Computing). Por isso, dada a já larga, e ainda crescente, utilização do Windows, decidiu-se implementar a plataforma PIPDS em tal ambiente. Por outro lado, para o osciloscópio e o analisador de espectros, foi criada uma segunda interface, denominada Analisador de Sinais, que utiliza a capacidade de temporização do Windows /Pet93/.

Quando se fala em desenvolver um aplicativo para rodar em um sistema operacional, normalmente não há distinção entre o sistema operacional (OS) e a Interface de Programação de Aplicativo ou API. Uma API é um conjunto de regras para se ter acesso a algum conjunto de serviços. As regras incluem funções, tipos de dados e quaisquer considerações especiais sobre a obtenção dos serviços.

Um sistema operacional é um produto que se adquire. Adquire-se um sistema operacional porque ele suporta uma API necessária para rodar certo conjunto de aplicativos. Como alguns sistemas operacionais suportam várias APIs, é importante distinguir entre um sistema operacional e as APIs suportadas. A Tabela 4.1 lista vários sistemas operacionais, assim como as APIs que cada um deles suporta.

Sistema Operacional	API(s) suportada(s)
MS-DOS	MS-DOS
Windows 3.1	MS-DOS, Win16
Windows 3.1 c/ Win32s	MS-DOS, Win16, Win32s
OS/2 2.0	MS-DOS, Win16, OS/2 caractere, OS/2 Presentation Manager
Windows NT	MS-DOS, Win16, Win32, POSIX, OS/2 baseado em caractere

Tabela 4.1. Sistemas Operacionais e APIs suportadas.

A API Win16 foi a principal interface de programação do Windows versão 1.x até 3.x. A Win16 baseia-se na arquitetura de 16 bits da família de microprocessadores Intel-86. Os membros de 16 bits dessa família têm suas raízes no aparecimento do Intel 8088, em 1978.

A Microsoft projetou a API Win32 como sucessora da Win16. O principal objetivo da Win32 é a compatibilidade retroativa com a Win16. Outro objetivo é a conversibilidade para plataformas não-Intel.

Então, qual a diferença entre as duas APIs? A diferença mais evidente é que as funções e as estruturas de dados da Win16 utilizam valores de 16 bits, enquanto as funções e estruturas de dados correspondentes da Win32 usam valores de 32 bits. Entretanto, os criadores da API Win16 anteciparam essa ampliação fornecendo um conjunto de tipos de dados que torna essa mudança quase invisível. O código Win16 que utiliza os tipos de dados portáteis é facilmente convertido para a Win32.

A API Win32 foi criada para explorar o endereçamento e os registradores de 32 bits dos processadores atuais. Essa API foi projetada para ser portátil e está disponível rodando sob o Windows NT nas seguintes plataformas:

Intel x-86 (80386 de 32 bits e superior)
MIPS R4000
DEC Alpha
Motorola PowerPC
Intergraph

Na programação da plataforma PIPDS adotou-se a API Win16, podendo futuramente migrar para uma API Win32, sem muita dificuldade.

4.3. A perspectiva do usuário da plataforma PIPDS

Em relação ao ambiente convencional do MS-DOS uma plataforma baseada no Windows tem consideráveis vantagens para o usuário e para o programador. Todos os programas Windows, inclusive a plataforma PIPDS, têm a mesma aparência e modo de operação fundamental: os usuários não precisam mais gastar um longo período de tempo aprendendo a usar o computador ou a dominar este novo programa. O programa ocupa uma janela - uma área retangular da tela - e é identificado por uma barra de título. A

maioria das funções do programa são iniciadas por meio do menu do programa. A Figura 4.1 mostra uma tela típica do programa PIPDS, com os vários componentes da janela identificados.

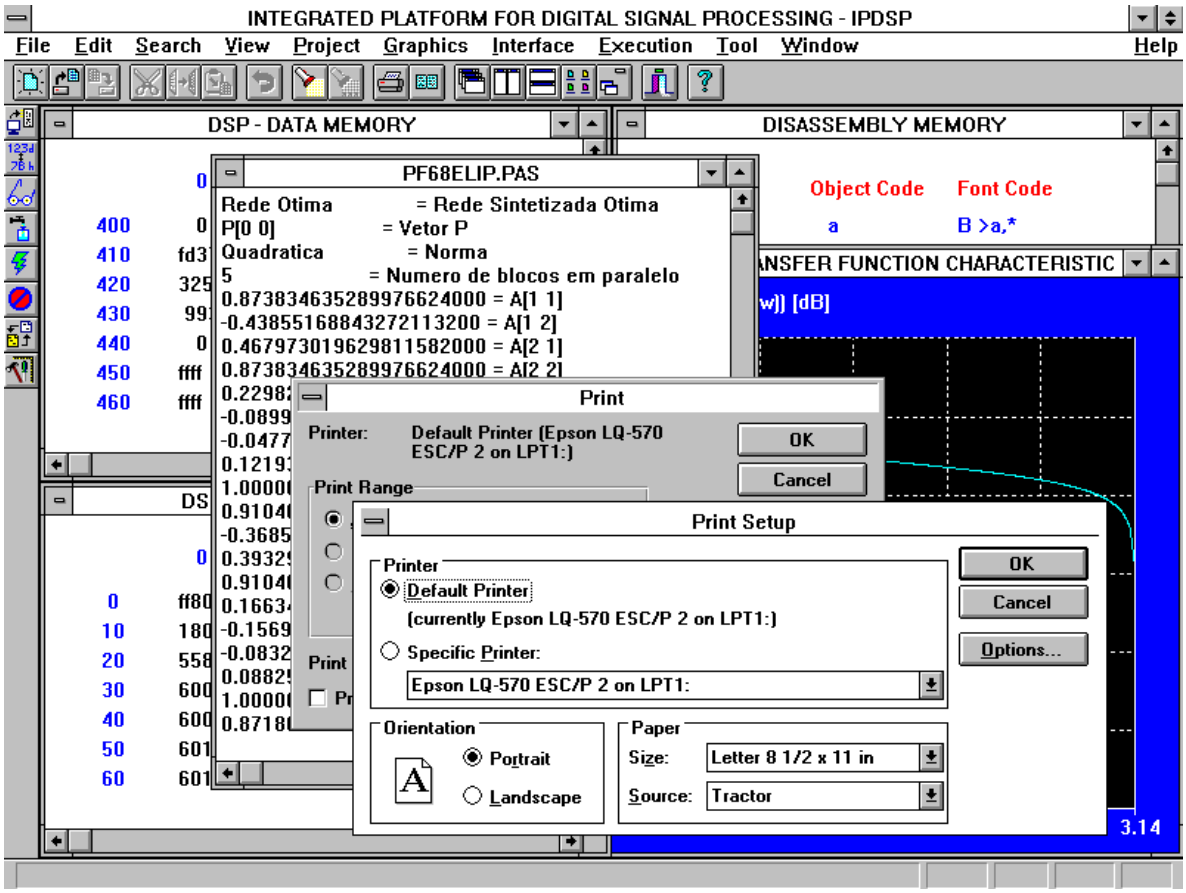


Figura 4.1. Aplicativo PIPDS rodando sob o Microsoft Windows.

O usuário também pode inserir informações adicionais, após alguns itens do menu invocarem caixas de diálogo. Uma caixa de diálogo encontrada em quase todo programa Windows é uma que abre um arquivo, a qual pode ser vista na Figura 4.2. Essa caixa de diálogo tem o mesmo aspecto (ou muito similar) em diferentes programas Windows, e quase sempre é invocada com a mesma opção de menu.

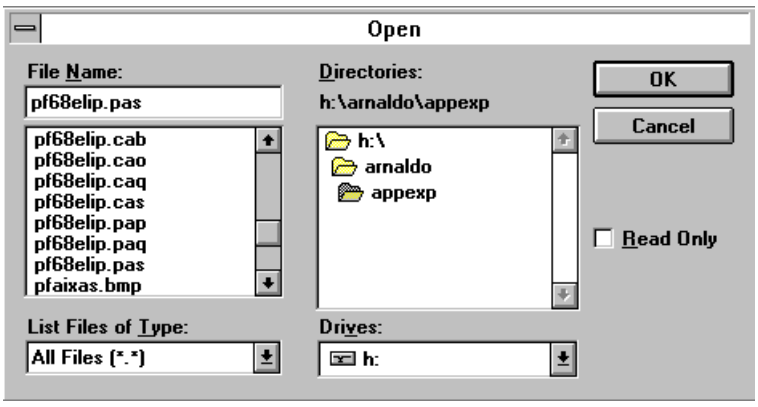


Figura 4.2. Caixa de diálogo “default” para abrir arquivos no Microsoft Windows.

Uma interface do usuário consistente possibilita, na perspectiva do programador, o uso de rotinas internas do Windows para a criação de menus e caixas de diálogo. Todos os menus têm a mesma interface de mouse e teclado porque o próprio Windows, e não o programa aplicativo, cuida dessa tarefa.

Por outro lado, o Windows permite a estruturação do sistema em um ambiente multitarefas. Apesar de algumas pessoas continuarem a questionar se a multitarefa é realmente necessária em um computador monousuário, os usuários estão definitivamente prontos para utilizá-la e podem beneficiar-se dela. A popularidade de programas residentes para o MS-DOS, como o Sidekick, prova isto. Embora, rigorosamente falando, programas residentes não sejam programas multitarefa, eles permitem uma rápida mudança de contexto, e isso envolve muito dos conceitos de multitarefa.

Todo programa, sob o Windows, torna-se na verdade, residente em RAM. Como mostra a Figura 4.3, muitos programas Windows, como por exemplo a plataforma PIPDS e o Analisador de Sinais, podem ser mostrados e executados ao mesmo tempo, cada um deles ocupando uma janela retangular na tela. O usuário pode mover as janelas na tela, alterar o seu tamanho, alternar as janelas e transferir dados de um programa para outro.

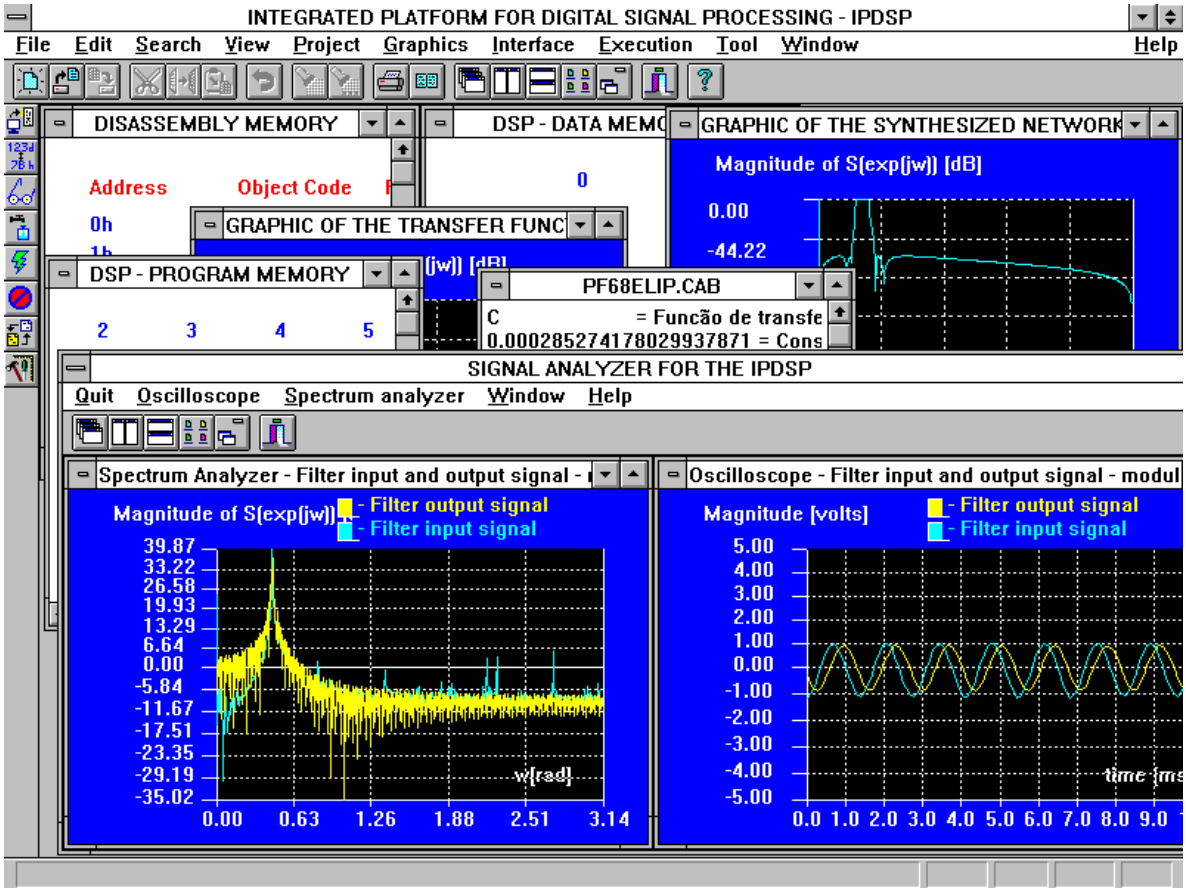


Figura 4.3. Diversas telas da plataforma PIPDS e do Analisador de Sinais.

A plataforma PIPDS e o Analisador de Sinais a ela associado podem utilizar gráficos e texto formatado, tanto no vídeo como na impressora, pois o Windows é uma interface gráfica. Uma interface gráfica não é somente mais atraente em aparência, mas também pode transmitir um grande número de informações ao usuário.

Como a impressora e o monitor de vídeo, esses programas feitos para o Windows não acessam diretamente o hardware dos dispositivos gráficos. O Windows contém uma linguagem de gráficos (chamada de Graphics Device Interface, ou GDI) que permite a fácil apresentação de gráficos e texto formatado. Um programa tal como o Analisador de Sinais, feito para o ambiente Windows, funcionará com qualquer placa de vídeo e qualquer impressora para a qual exista um controlador de dispositivo disponível para o Windows. O programa nem precisa determinar que tipo de dispositivo está conectado ao sistema.

Não foi uma tarefa fácil para os criadores do Windows colocar uma interface gráfica independente do dispositivo no IBM-PC. O projeto do PC baseou-se no princípio da arquitetura aberta: os fabricantes de hardware foram encorajados a desenvolver periféricos para o PC e o fizeram em grande número. Embora diversos padrões tenham emergido, os programas convencionais para o MS-DOS precisam individualmente suportar diferentes configurações de hardware. Por exemplo, é bastante comum um programa de processamento de texto ser vendido com um ou dois disquetes contendo pequenos arquivos, um para cada tipo de impressora.

Devido ao suporte ser responsabilidade do próprio Windows, os programas voltados para este ambiente não requerem esses controladores. Isso beneficia os usuários pois facilita a instalação da maioria dos programas. Tudo o que o programa necessita pode ser incluído em um único arquivo .EXE. Geralmente, o usuário somente precisa copiar o arquivo .EXE para o disco rígido, carregar o Windows e executar o novo programa.

4.4 Desafios da programação para o ambiente Windows

A maioria dos programadores está acostumada a escrever código sequencial e orientado por procedimentos. Esse tipo de programa tem início, meio e fim bem definidos. Considere-se, por exemplo, um programa que exiba uma série de telas de entrada de dados para a criação de um gráfico, que pode ser a função de transferência do filtro ou a sua resposta ao impulso. O fluxograma da Figura 4.4. ilustra uma sequência rigorosa na qual esse programa poderia operar.

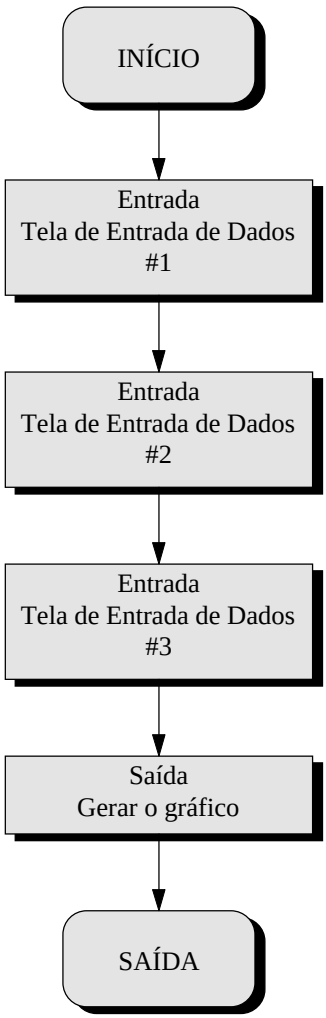


Figura 4.4. Um trecho de programa orientado sequencialmente.

Para esta discussão, esse fluxograma pode representar um programa utilizado por um projetista de filtros digitais. A primeira tela de entrada aceita informações do tipo de filtro a ser projetado: passa baixas, passa-altas, passa-faixa, etc. A segunda tela possibilita a síntese do filtro em redes representadas no espaço de estados. E, finalmente, a terceira tela gera os gráficos solicitados. Cada tela de entrada deve ser preenchida corretamente, antes que o projetista de filtros possa prosseguir, e as três telas devem ter informações corretas para que se possa gerar o gráfico corretamente.

À primeira vista, esse parece um modo razoável de processar o programa. Afinal, a tarefa de um programa de computador não é apenas emitir gráficos, mas verificar se as informações corretas foram recebidas. Contudo, existem limitações nessa abordagem, que são resultado direto dessa orientação sequencial. Por exemplo, como o programa dita a sequência de operação, o projetista de filtros não pode ir à segunda tela - de síntese de filtros - sem primeiro introduzir todas as informações do filtro a ser projetado. O resultado é a geração de passos desnecessários.

Embora o programa garanta que todas as informações exigidas tenham sido introduzidas, ele não leva em conta as exceções do mundo real. Por exemplo, se o projetista fosse gerar os gráficos de vários filtros, ele precisaria percorrer as três telas para cada filtro, para ter seus gráficos mostrados. Mais uma vez, o programa faz sua parte, garantindo que todas as informações necessárias sejam coletadas, mas ao custo de um trabalho adicional por parte do projetista de filtros.

Por outro lado, um programa orientado por eventos permite ao projetista introduzir os dados na ordem que lhe parecer adequada. Talvez ele escolhesse a mesma ordem imposta pelo programa sequencial. Contudo, ficaria livre para executar as tarefas necessárias na sequência que satisfizesse as exigências para cada filtro. A Figura 4.5 dá uma idéia de como a abordagem orientada por eventos poderia alterar o programa sequencial tradicional, descrito anteriormente.

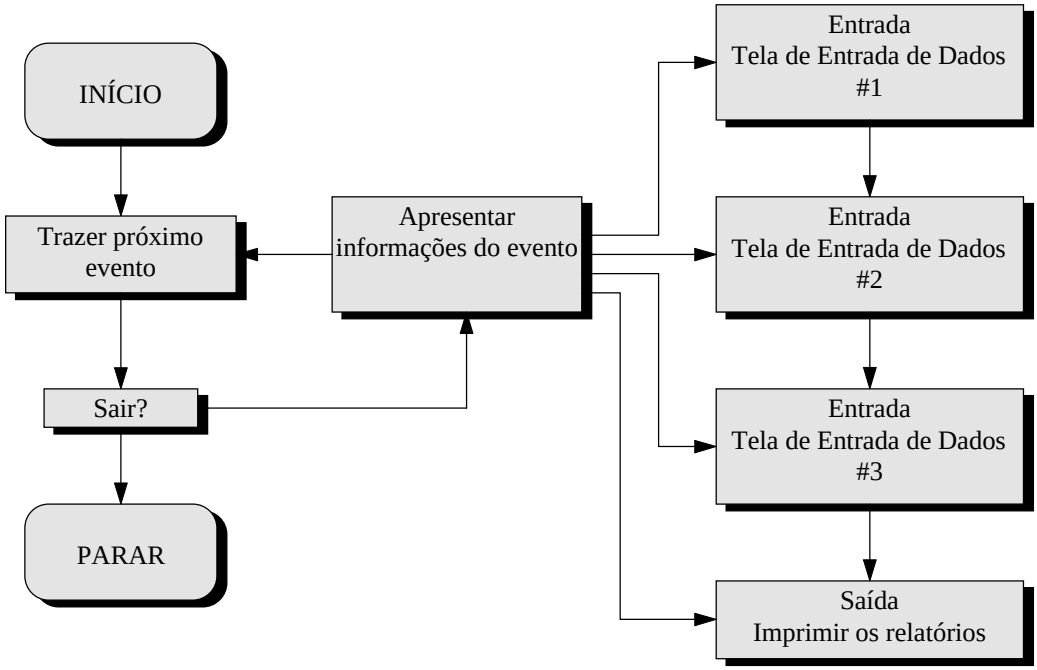


Figura 4.5. Um programa orientado por eventos.

Porém, esse é apenas um aspecto pelo qual um programa orientado por eventos difere de um sequencial. Um sistema operacional orientado por eventos, como o Windows, vai ainda mais longe, para que, por exemplo dentro das telas de entrada de dados, o projetista de filtros tenha enorme flexibilidade na ordem em que os campos são preenchidos.

Um programa sequencial é construído em um conjunto mal organizado de modos. Um modo é o estado de um programa no qual as ações do usuário são interpretadas de uma maneira específica e produzem um conjunto específico de resultados. Em uma reação contra os programas orientados sequencialmente, alguns programadores de GUI podem dizer que os modos são ruins. Infelizmente, isso é uma simplificação.

Um grande problema relacionado aos modos ocorre quando o usuário não pode ir de um modo para outro facilmente. Em nosso programa sequencial de geração de gráficos, por exemplo, cada um dos quatro passos é um modo. Mas, como o programa impõe que o usuário percorra os modos em uma sequência rígida, ele é impedido de estruturar a utilização do programa para satisfazer às várias solicitações feitas.

Outro problema com os modos ocorre em programas que dependem de que o usuário lembre o modo corrente. Em vez disso, o programa deve oferecer pistas visuais para ajudar ao usuário a identificar o modo corrente. No Windows, existem muitos objetos de interface de usuário que suportam isso. O formato do cursor do mouse, por exemplo, pode indicar quando um pacote de desenho está no modo de desenho de retângulo e quando está no modo de desenho de texto. As caixas de diálogo modais, por exemplo, são um modo muito comum de receber entrada do usuário, necessária para completar um comando.

Os modos de um programa devem ser cuidadosamente planejados para evitar a perda de dados, quando o usuário vai para um outro modo acidentalmente. Do ponto de vista da programação, os programas modais são mais fáceis de serem implementados do que os que não têm modo. O código que suporta cada modo pode ser escrito e depurado em relativo isolamento das outras partes do programa. Porém, o Windows facilita a criação de programas sem modo, pois toda a interação com o mundo exterior - todos os eventos - é direcionada a um programa assim. Todos os eventos de interesse geram mensagens. Elas são informações sobre alguma alteração na interface de usuário, como uma janela sendo movida ou uma tecla sendo pressionada. As mensagens notificam a um programa que um temporizador foi acionado. As mensagens são utilizadas para operações de compartilhamento de dados.

Do ponto de vista da programação, uma mensagem é um valor no qual, para facilidade de leitura, é atribuída uma constante simbólica que começa com as letras WM_(abreviatura de “Windows Message”). Por exemplo, a mensagem `WM_BUTTONDOWN` informa ao programa que o usuário pressionou o botão esquerdo do mouse. Outra mensagem, que é enviada depois que o botão esquerdo do mouse é solto, é `WM_LBUTTONUP`. Em todo este trabalho são usadas mensagens, no contexto de diferentes assuntos abordados.

As mensagens são muito importantes para quem programa para o Windows. A maior parte do trabalho realizado como programador envolve decidir quais mensagens devem ser processadas e quais devem ser ignoradas. Uma coisa que se deve ter em mente é que as mensagens não aparecem em nenhuma ordem predefinida.

A orientação a mensagens do Windows é muito conveniente para programas que exigem um alto nível de interação com o usuário, como é o caso da plataforma PIPDS e do Analisador de Sinais, e por isso ela foi aqui utilizada.

Um sistema operacional orientado por eventos, como o Windows, coloca alta prioridade na permissão ao usuário para intervir em qualquer ponto de um processo. Por outro lado, um programa orientado sequencialmente coloca alta prioridade na imposição

da sequência em que a tarefa deve ser executada. Em um programa orientado sequencialmente, todas as tentativas são feitas para que o programador crie regras arbitrárias a respeito da ordem em que os passos devem ser executados. Embora seja possível criar programas orientados sequencialmente no Windows, o esforço extra que isso requer praticamente garante que essas restrições só estejam em vigor onde forem realmente necessárias.

Uma outra diferença é que toda saída gerada pelo Windows é gráfica. As formas geométricas são mais fáceis, pois o programa não precisa calcular cada pixel. Pela simples chamada da rotina `Rectangle`, por exemplo, a GDI desenha um retângulo preenchido. A saída de texto é mais difícil, pois a orientação gráfica da GDI exige que se trate o texto como um objeto gráfico. O texto é posicionado usando-se coordenadas de pixel no lugar de coordenadas de célula.

A GDI oferece gráficos independentes do dispositivo. Isso significa que um programa Windows pode desenhar em qualquer dispositivo usando o mesmo conjunto de funções. Por exemplo, a rotina `Rectangle` é chamada para desenhar retângulos no vídeo e também em impressoras. A GDI esforça-se para que, do ponto de vista do programa, todos os dispositivos pareçam semelhantes. Isso inclui os dispositivos que só sabem ligar e desligar pixels - como a placa de vídeo CGA -, bem como dispostivos que conseguem fazer desenhos complexos, como as impressoras PostScript. Cada dispositivo tem um controlador responsável por fazer o desenho real.

A GDI conhece quatro tipos de dispositivos: o vídeo, dispositivos de impressão (como impressoras e plotadoras), mapas de bits e metarquivos. Dois deles são dispositivos físicos: o vídeo e os dispositivos de impressão. Os outros dois, mapas de bits e metarquivos, são pseudodispositivos. Um pseudodispositivo fornece um meio de armazenar uma figura na RAM ou no disco, assim como um modo padrão de compartilhar imagens gráficas entre os aplicativos.

Ao desenhar no vídeo, a GDI fornece gráficos orientados para a janela. Isso significa que cada janela é tratada como uma superfície de desenho separada. Quando um programa desenha em uma janela, as coordenadas de desenho padrão são definidas de modo que a origem (0, 0) esteja no canto superior esquerdo da área cliente da janela.

Os gráficos orientados para janela também significam que os desenhos são recortados automaticamente em uma janela. O recorte significa que o desenho feito para cada janela está limitado às suas margens. Mesmo que uma janela tentasse desenhar além de suas margens, ela não poderia. Os pixels de uma janela são automaticamente protegidos da manipulação por outras janelas. Esse mecanismo de proteção funciona em mão dupla, de modo que, quando se desenha, não é preciso preocupar-se com sobrescrita na janela de outro programa. Claramente, isso é vantajoso, no caso da plataforma PIPDS e do seu Analisador de Sinais, em que vários gráficos podem ser gerados: cada um deles ficará automaticamente limitado à sua janela.

Outro aspecto interessante é que o Windows tem suporte interno para vários objetos de interface do usuário: janelas, ícones, menus, caixas de diálogo, etc. Suporte interno significa que o esforço exigido para se criar e manter esses objetos é mínimo.

Dentre os objetos de interface de usuário, o mais importante é a janela. Qualquer programa que pretenda interagir com o usuário precisa de uma janela, pois ela recebe entrada de mouse e teclado e exibe a saída do programa. Todos os outros objetos de interface de usuário, como menus, barras de rolagem e cursores executam tarefas de suporte a partir dessa base.

4.5. Ferramentas usuais disponíveis para a programação Windows

No desenvolvimento da plataforma PIPDS, estudou-se as principais rotinas desempenhadas pelo projetista de filtros digitais. De posse do fluxograma mostrado pela Figura 1.3, ficou claro que seria necessário um código em Windows bastante elaborado.

O Windows oferece mais de 600 funções através de sua Interface de Programa Aplicativo. Achou-se que a biblioteca ObjectWindows simplificaria bastante o tratamento com a API, encapsulando com maior simplicidade muitos recursos complexos do Windows. Escrever software para Windows diretamente em C ou em linguagem assembly é possível, mas a qual custo? A solução para a maioria dos projetistas de aplicativos para o ambiente Windows é poupar tempo utilizando bibliotecas e sistemas de desenvolvimento de outros fabricantes para criar aplicativos. A questão é: quais ferramentas disponíveis utilizar? Um pequeno conjunto de soluções é apresentado a seguir, assim como é indicada a solução adotada.

4.5.1. A aproximação “straight C”

O “straight C” nunca foi projetado para manipular o altamente complexo mundo da programação para Windows, apesar de os experts em informática afirmarem que C e Windows combinam bastante. A maioria dos projetistas considera que se leva muito tempo para escrever e depurar um código confiável escrito em C.

4.5.2. A aproximação geradora de aplicativos

Para poupar tempo, alguns projetistas optam por um Gerador de Aplicativos. Geralmente esses produtos criam arquivos de programa compostos de códigos-fonte a partir de comandos, diálogos, controles e outras opções de menu selecionados. Quando o “shell” do arquivo-fonte é finalizado, o projetista preenche os campos em branco com o código do aplicativo. Um gerador de aplicativo é uma maneira prática de criar protótipos, telas de entrada para bancos de dados e aplicativos similares. Todos os geradores, entretanto, possuem duas falhas principais: em primeiro lugar, ainda tem-se que escrever a parte principal do programa porque nenhum desses geradores de aplicativos é capaz de fazê-lo; em segundo lugar, quando se necessita fazer o “upgrade” do código, esse gerador pode não ser útil.

4.5.3. A aproximação “plug-and-play” visual

A aproximação através da programação visual é altamente recomendada para alguns tipos de aplicativos. Produtos como o Visual Basic e o ToolBook oferecem objetos do tipo plug-and-play, que interpretam instruções quando selecionados ou são vinculados de alguma forma a outros objetos e a operações pré-programadas.

O Visual Basic é adequado para telas de entrada de bancos de dados, calendários e utilitários similares de pequena escala. Para o desenvolvimento de aplicativos de uso geral, entretanto, a aproximação plug-and-play não é a mais apropriada. Quando se tem que considerar a performance e quando não se pode ficar restrito ao conjunto limitado de recursos oferecidos em um sistema de programação visual, é preciso usar ferramentas poderosas e uma linguagem para desenvolvimento de uso geral, como por exemplo C ou C++.

4.5.4. Aproximação através da biblioteca de classes

Essa aproximação conduz ao que é, provavelmente, a melhor aproximação para a programação Windows - a utilização do C++ e de uma biblioteca de classes, como por exemplo a "ObjectWindows Library (OWL)" da Borland International, Inc. Essa solução oferece aos programadores duas vantagens importantes sobre os métodos anteriores:

- as classes do C++ modelam a arquitetura do Windows muito melhor do que a aproximação em C. Uma classe de janela, por exemplo, incorpora dados e funções. A classe automaticamente cuida das necessidades internas, que devem ser fornecidas explicitamente em C. Além do encapsulamento, o uso de tabelas de respostas e o uso de funções-membro da classe podem responder diretamente a mensagens do Windows, simplificando a difícil tarefa de escrever um código para um sistema operacional orientado a eventos;

- uma biblioteca de classes como a OWL fornece uma estrutura de aplicativo na qual se montam os programas. Através da herança deriva-se novas classes baseadas naquelas que o Borland C++ fornece, para que se possa acrescentar os recursos necessários. Com o C++ reutiliza-se o código já existente, o que poupa tempo de desenvolvimento, minimiza a depuração e aumenta a confiabilidade do aplicativo.

A OWL não é a única biblioteca de classes para Windows no mercado. Alguns produtos como o Microsoft Foundation Class Library fornecem classes do tipo "wrapper" que atuam como residentes para funções do Windows. As bibliotecas dessas funções "wrapper" tornam possível utilizar o C++ em vez do straight C para programação Windows. A OWL possui classes "wrapper", além de fornecer também uma estrutura que ajuda no projeto de direcionamento, molde e formato do aplicativo.

Outros produtos baseados no ambiente Windows oferecem compilação de fonte única para diferentes sistemas operacionais. Pode-se comprar uma biblioteca, por exemplo, com classes idênticas para os diversos sistemas operacionais. Infelizmente, paga-se um alto preço em performance perdida para esse nível de compatibilidade. As funções em bibliotecas de multiplataformas precisam ser traduzidas para chamadas de sub-rotinas de baixo nível para o Macintosh e para o Windows, o que gera perda de tempo. A OWL realiza chamadas diretas para a API (Interface de Programas Aplicativos) do Windows, muitas das quais sendo compiladas como declarações de

linha - em outras palavras, não como sub-rotinas aninhadas. Devido ao uso extensivo de códigos em linha pelo Borland C++, o uso da biblioteca não adiciona praticamente nenhum "overhead" apreciável ao tempo de execução do aplicativo final. Muitos programadores experientes insistem nisso porque uma biblioteca de classes ajuda a organizar programas de forma mais eficiente e o programa final geralmente trabalha melhor que o código direto escrito a partir de tentativas manuscritas. Essa afirmação é difícil de ser provada; entretanto, aqueles que são familiares ao OWL geralmente confirmam que o uso de uma biblioteca de classes não necessariamente reduz a performance, o que pode ser esperado com linguagens interpretadas como o Visual Basic.

Por essas vantagens, decidiu-se usar esta solução, tanto na implementação da plataforma PIPDS quanto do Analisador de Sinais a ela associada.

4.6. A biblioteca de classes "ObjectWindows Libray" - OWL

No projeto da Plataforma Integrada para Processamento Digital de Sinais, baseado em objetos, as seguintes questões precisaram ser respondidas:

- que classes seriam implementadas?
- que estrutura de dados seria empregada para cada classe?
- que operações seriam oferecidadas por cada classe e quais seriam seus métodos?
- como seria implementada a herança de classe e como ela afetaria os dados e os métodos?
- que interface com o usuário seria necessária?

A programação procedimental estimula a visão global: pode-se acessar qualquer dado e estabelecer uma ligação com qualquer sub-rotina. A programação baseada em objetos, por outro lado, tem uma perspectiva estritamente local: os dados e o código para uma classe são encapsulados, o que é essencial para decidir que classes têm quais responsabilidades e, depois, encapsular essas responsabilidades.

Entre as muitas classes da OWL pode-se destacar aquelas abaixo, que foram amplamente utilizadas no desenvolvimento da plataforma:

Janelas: a classe `TWindow` forneceu uma interface de classe básica para elementos de janela como caixas de diálogo, controles, janelas-filhas e outros;

Classes de aplicativo e de módulo: a classe `TModule` formou a base para a classe `TApplication`, que incorporou as funções globais do aplicativo, como por exemplo tarefas de inicialização, loops de mensagem e manipulação de erros.

Janelas MDI: duas classes suportaram inteiramente a MDI (Multiple Document Interface). São elas: `TMDIClient` e `TMDIChild`. As janelas MDI-filhas foram instâncias da classe `TMDIChild`. Usou-se abundamente as propriedades destas duas classes na implementação da plataforma PIPDS, conforme será descrito detalhadamente mais adiante;

Classes gráficas: toda a GDI do Windows foi incorporada em um conjunto de classes dependente do contexto do dispositivo, como por exemplo `TClientDC`, `TPrintDC`, `TPaintDC` e outras: para desenhar em uma janela ou para imprimir texto e gráficos, simplesmente gerou-se um objeto desse tipo e chamou-se uma função-membro em referência a esse objeto;

Classes de objeto GDI: todos os objetos, como canetas, pincéis, paletas de cores, fontes, ícones, cursores, mapas de bits e outros, possuiram classes GDI associadas. Uma caneta do Windows foi um objeto da classe `TPen`; um pincel foi uma instância `TBrush` e assim sucessivamente. Para utilizar esses itens, primeiro eles foram selecionados para um objeto do tipo "device context". Em seguida, chamou-se as funções-membro da classe, que geraram gráficos na saída utilizando as ferramentas selecionadas. Como o objeto "device context" conhece os objetos gráficos que possui, o contexto automaticamente deletou os objetos quando se terminou de utilizá-los, um recurso que ajudou a reduzir os problemas de memória causados pela falha na deleção de canetas e pincéis - problemas comuns encontrados em programas convencionais do Windows;

Janelas Decoradas: essas classes simplificaram a programação de barras de ferramentas, linhas de status e outras decorações de janelas. Definiu-se o layout de janela com `TLayoutWindow` e `TLayoutMetrics`. Inseriu-se janelas-cliente com ajuste próprio utilizando `TDecoratedFrame` e `TDecoratedMDIFrame`. Acrescentou-se elementos no formato "tile" com `TGadgetWindow` e `TGadget`. Todas essas classes definiram janelas com recursos que poderão ser apreciados por usuários experientes;

Caixas de diálogo: a classe `TDialog` gerou interfaces com diálogos do Windows projetados como recursos através do Resource Workshop e em arquivos de "script" de recursos. O mecanismo para transferência de dados do `TDialog` facilitou a cópia de informações de e para controles de diálogo;

Diálogos comuns: essas classes forneceram interfaces para todos os diálogos comuns do Windows. Incluiram-se elementos para escolha de arquivo (`TOpenSaveDialog`, `TFileOpenDialog` e `TFileSaveDialog`), uma interface para opções de impressão (`TPrintDialog`) e elementos geradores de "prompt" para busca e substituição (`TFindReplaceDialog`, `TFindDialog` e `TReplaceDialog`);

Classes de controle: a classe `TControl` gerou objetos de controle em janelas. A OWL ofereceu três tipos de controle: controles padrões do Windows, Widgets (geradores de slides personalizados, por exemplo, escritos inteiramente em C++) e controles de decoração (barras de ferramentas e linhas de status). O OWL também suportou controles do Borland Windows Custom Controls (BWCC) assim como controles do Microsoft 3-D;

Classes de impressão: as classes `TPrinter` e `TPrintOut` foram usadas para se permitir realizar impressão simples e de várias páginas;

Classes de documento e de visualização: as classes `TDocManager`, `TDocument` e `TView` forneceram um modelo para visualização de documentos que molda os dados para propósitos de entrada e saída. Simplificando, essas classes tornaram possível projetar serviços de E/S que funcionaram independentemente da informação acessada.

Outras classes utilizadas: a OWL está repleta de outras classes que foram abundantemente utilizadas. `TRect` e `TPoint`, por exemplo, simplificaram a manipulação de coordenadas de tela, as classes de menu ajudaram a criar menus dinâmicos e "popup", a classe Clipboard simplificou a passagem de informações de e para o "clipboard" do Windows, e as classes de validação facilitaram a solicitação e a verificação dos dados formatados.

4.7. Descrição da interface implementada

Antes de descrever a plataforma PIPDS de maneira mais precisa, é importante destacar sua estrutura modular: ela é implementada através de um programa principal (núcleo central) que gerencia diversos módulos, os quais, por sua vez, possuem sub-módulos. São onze os módulos principais: Arquivo, Edição, Procurar, Projeto, Gráficos, Interface, Execução, Ferramenta, Visualizar, Janela e Help.

A utilização de tal estrutura também é parte da decisão de produzir uma interface extremamente amigável para o usuário, a começar por uma barra de menu para a seleção do módulo desejado. Claramente, porém, a seleção do módulo na barra de menu não é arbitrária, já que alguns deles podem requerer que outros tenham sido previamente utilizados (exemplo: não se pode carregar um conjunto de coeficientes na memória de dados do DSP sem antes carregar o código objeto correspondente em sua memória de programa e sem antes sintetizar um filtro qualquer).

A linguagem utilizada em sua implementação foi o BORLAND C++ 4.02 (incluindo o conjunto de bibliotecas OWL 2.0), com recursos de programação orientada a objetos, tendo sido feita a opção pelo ambiente MS-WINDOWS for WORKGROUPS 3.11, a versão para rede padrão Windows 3.1, de modo a facilitar a criação da interface homem-máquina, o gerenciamento de múltiplas tarefas e a comunicação entre as mesmas (através do uso de mensagens). Um outro fator importante a ser destacado, que é conseqüência do uso do Windows, é o fato da arquitetura implementada ser totalmente independente dos dispositivos periféricos, ainda que permanecendo limitada ao mundo dos PC's e seus compatíveis /Pet93/. A Figura 4.6 ilustra a plataforma PIPDS.

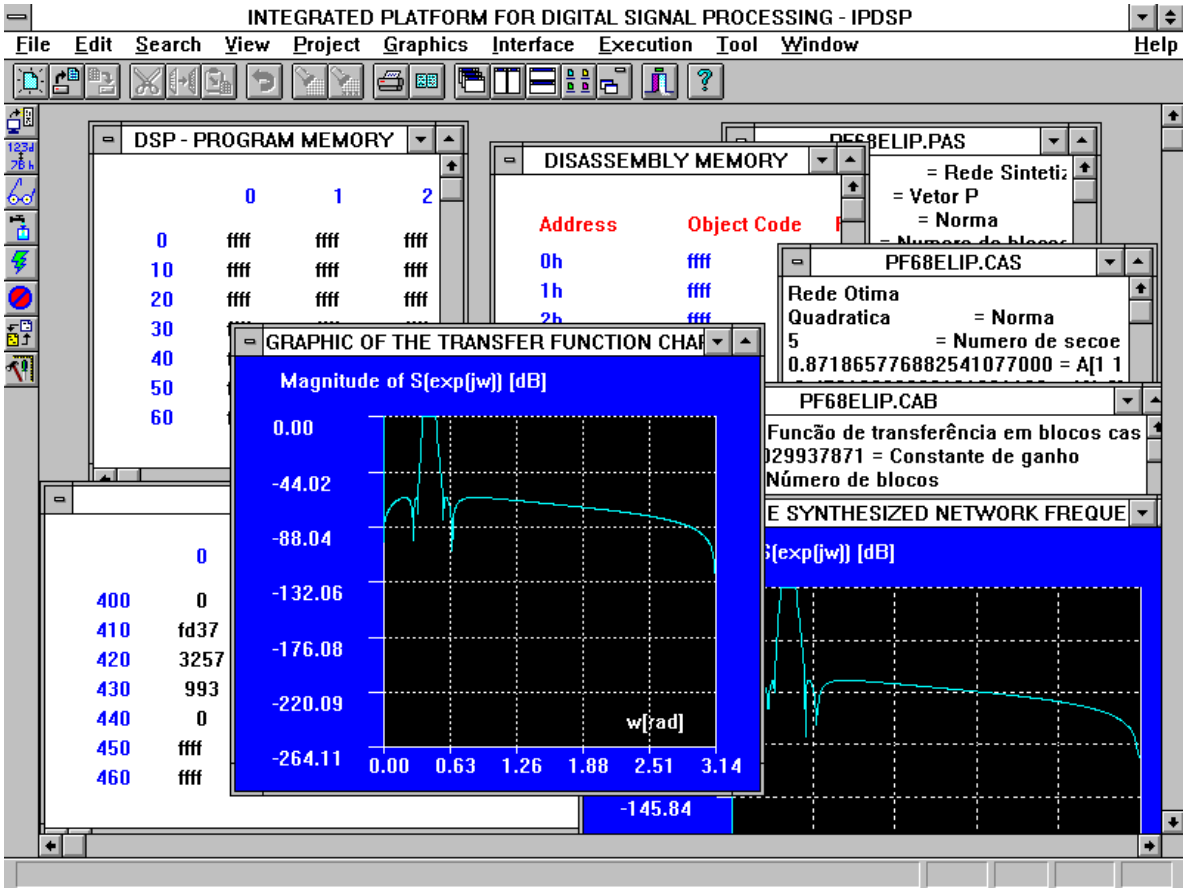


Figura 4.6. A plataforma PIPDS.

Para um melhor acompanhamento da descrição a seguir, seria bastante oportuno o acesso à plataforma descrita, para que se possa verificar os passos e objetos descritos “on-line”.

Dentre os módulos implementados, o módulo projeto contém as diversas rotinas utilizadas para o projeto de filtros digitais. Os sub-módulos existentes são:

- aproximação da função de transferência: em que é obtida a função de transferência correspondente a um gabarito de atenuação especificado nas formas polinomial distribuída e cascata de blocos quadráticos;

- síntese dos filtros: em que são obtidos os coeficientes correspondentes às estruturas usadas para realizar o filtro com precisão infinita /Sim94/, nas formas cascata (onde também se pode efetuar a ordenação das seções, antes da síntese) ou paralela. Deve ser destacado que, no momento, apenas quatro estruturas no espaço de estados /Sim94/ são manipuladas em tal módulo, mas ele pode ser ampliado para contemplar outras classes de estruturas;

- quantização das estruturas sintetizadas: em que as redes sintetizadas em precisão infinita são então quantizadas para um número finito de bits.

Este módulo disponibiliza ao usuário caixas de diálogo que o conduzem na escolha do tipo de filtro que deseja projetar. A primeira caixa de diálogo informa quais os tipos de filtro disponíveis na plataforma, para se realizar a aproximação.

Ao ser escolhido o tipo de filtro a ser aproximado, uma nova caixa de diálogo permite que o usuário escolha o tipo de aproximação desejado. Os tipos de aproximação Butterworth, Chebyshev e Elíptico estão disponíveis na plataforma. É possível, caso o tipo de filtro escolhido for incorreto, retornar através deste diálogo ao diálogo anterior.

Logo em seguida, um novo diálogo, agora solicitando que sejam fornecidos os valores para o gabarito de atenuação e a autorização para a geração do filtro, é mostrado pela plataforma. O usuário pode informar que deseja entrar com os valores escolhendo a opção correspondente. Caso o usuário solicite que seja gerado um filtro antes de preencher seu gabarito de atenuação, uma mensagem será mostrada, indicando que está incompleta a especificação da magnitude e o diálogo solicitando a especificação do filtro é novamente mostrado.

É importante destacar que todos os diálogos mostrados possuem Help On-Line, ou seja, caso o usuário tenha alguma dúvida na operação da caixa de diálogo, basta clicar no botão indicando a ajuda, e uma tela de Ajuda sobre aquele tópico estará disponível.

Quando o diálogo para a especificação do filtro escolhido estiver disponível, vários campos devem ser preenchidos pelo usuário para posteriormente serem validados. Esses campos vão desde a especificação da freqüência de amostragem até a especificação dos limites da banda passante e de rejeição do filtro, com as correspondentes atenuações desejadas, em decibéis.

Caso a especificação da atenuação mínima em decibéis não seja fornecida pelo usuário, será solicitado a ordem do filtro desejado. Somente assim o filtro então poderá ser gerado. Existe ainda a opção do usuário abortar o processo não autorizando a geração do filtro.

Após o filtro ter sido gerado um novo diálogo é mostrado e o usuário pode escolher que tipo de configuração da função de transferência para o filtro ele deseja. Estão disponíveis as formas polinomial distribuída e cascata de blocos de segunda ordem. Escolhendo uma das formas disponíveis, a plataforma abre uma janela de diálogo padrão do Windows que solicita que seja informado o nome do arquivo. Deve ser observado aqui que o nome do arquivo vem acompanhado de uma extensão já pré-definida pela plataforma, com o objetivo de caracterizar o tipo de função de transferência e facilitar quando o usuário desejar visualizar os coeficientes através do módulo de edição de arquivos ASCII.

O usuário desejando visualizar/editar os coeficientes da função de transferência obtida pelos procedimentos anteriores pode lançar mão do módulo de edição de arquivos, que consiste de ferramentas para a completa edição de arquivos ASCII. Entre as facilidades deste módulo, estão o acesso a qualquer parte da unidade de armazenamento de arquivos do IBM-PC, inclusive da rede, se a máquina estiver ligada em uma rede local. Arquivos de coeficientes podem ser editados usando ferramentas que

permitem entre outras atividades imprimir os coeficientes, podendo também visualizar a tela de impressão antes de mandar para o “spooler”. Existem ainda os módulos que permitem a edição de texto, onde o usuário pode, por exemplo procurar por uma palavra ou coeficiente específico. Pode ainda substituir a cadeia encontrada por uma outra cadeia de caracteres, entre outras tarefas tais como cortar, colar e copiar em outros lugares.

Neste ponto é importante discutir a implementação de alguns dos recursos descritos nos parágrafos anteriores, começando pelos diálogos.

Antigamente a maioria das interfaces de software tinham pouco mais do que uma série de “prompts” de texto que apareciam na tela de um terminal e pediam respostas dos usuários. Certas respostas levavam a outros “prompts”, atravessando um caminho tortuoso através da lógica do programa. Como as antigas respostas eram levadas para fora da tela, era difícil saber como voltar através das suas respostas prévias uma vez que tivesse se enredado na teia dos comandos e menus inter-relacionados do programa.

Como pode ser observado através do uso da plataforma PIPDS, sua programação utilizando um sistema operacional visual como o Windows colocou o usuário no controle das interfaces do software, tornando possível comutar tarefas de modo mais ou menos aleatório, como as pessoas estão acostumadas a fazer - isto é, na ordem que for mais atraente no momento. Não é sugerido que deva ser dado ao usuário um controle dispersivo sobre as tarefas de programação e implementação de um filtro, mas os elementos de interface, como as caixas de diálogo e os controles, ajudaram a tornar o software mais amigável e fácil de usar, principalmente por fornecer aos usuários uma grande parcela de autoridade sobre a operação do programa.

Um dos objetivos primários de uma interface Windows bem escrita é desenvolver programas que respondam às exigências do usuário. As caixas de diálogo foram ferramentas-chave para essa meta. Como foi mostrado anteriormente, uma caixa de diálogo forneceu um tipo de formulário eletrônico, no qual se selecionaram itens, entrou-se com valores e forneceram-se outras informações. As caixas de diálogo também puderam revelar informações importantes - qual seria o conteúdo do registros do DSP, por exemplo, ou a quantidade de ciclos necessários para executar um trecho de programa contido na memória do DSP - como será visto mais adiante. Como pode ser observado, com a operação da plataforma, muitas caixas de diálogo apareceram como resultado da seleção de comandos de menu que terminavam por reticências, assim como no comando Open... Em algumas ocasiões a reticências indicam que selecionar o comando abrirá uma caixa de diálogo. Quando o projetista de filtros vê um comando desse tipo, sabe que selecioná-lo dá acesso às escolhas adicionais, na forma de uma caixa de diálogo. Desse modo, as caixas de diálogo serviram como extensores do menu, ou sistemas de comando avançados, aos quais os usuários tiveram acesso através da barra de menu.

A seguir será mostrado como foi utilizada a classe `TDialog`, como foram projetados os recursos de caixa de diálogo, como foram projetados os diálogos modais e sem modo e como foram utilizadas as caixas de diálogo comuns do Windows.

Existem basicamente dois tipos de caixas de diálogo utilizados pela plataforma PIPDS: aquelas que o Windows (ou uma outra fonte) forneceu na forma pronta para uso e aquelas que foram criadas para o uso específico da plataforma. Os diálogos já prontos incluíram caixas de diálogo comuns do Windows, como a sempre presente File/Open... Em todos os casos, pôde-se programar interfaces para diálogos construindo objetos da classe `TDialog`.

Os diálogos são janelas especializadas e, portanto, a classe `TDialog` foi derivada naturalmene da `TWindow`. Um objeto diálogo (isto é, qualquer objeto da classe `TDialog` ou outra derivada) forneceu programas com uma interface bem-definida para o elemento interno de diálogo mantido pelo Windows. Melhor do que manipular diretamente cada elemento, como seria feito em um programa convencional em C, foi utilizar mais facilmente as funções-membro, os itens de dados e as tabelas de resposta da classe `TDialog` para realizar operações relacionadas com diálogos.

Essas operações, na plataforma, se ocuparam, em sua maioria, com o envio e recebimento de informações para os controles de diálogo. Para isso, `TDialog` forneceu um mecanismo de transferência de dados, o que simplificou em muito a inicialização dos valores de controle e o recebimento de entradas do usuário.

Programou-se diálogos modais e sem modo. Um diálogo modal coloca o programa no modo diálogo, o que requer que os usuários fechem a caixa de diálogo antes de selecionar outras operações do programa, como é o caso, por exemplo, quando o projetista precisa antes fechar a caixa de diálogo que seleciona o tipo de filtro a ser projetado para que possa acessar a próxima caixa de diálogo. Um diálogo sem modo opera mais como uma janela-filha (o termo janela-filha será explanado mais adiante). Assim como uma janela-filha, um diálogo sem modo é o mesmo que um modal, mas ele não proíbe os usuários de selecionar comandos do menu e realizar outras operações. Um diálogo File/Open... é um diálogo modal - deve-se selecionar um arquivo e fechar o diálogo antes de poder utilizar outros comandos do programa. Um diálogo de arquivo texto disponível na plataforma denominado Find... é um exemplo de diálogo implementado como um diálogo sem modo - neste diálogo pode-se entrar com os comandos ou digitar as alterações nos documentos enquanto o diálogo estiver ativo durante a procura por uma palavra.

As caixas de diálogo usadas pela plataforma foram janelas-filhas possuídas, normalmente, pela janela principal (a janela-mãe) do programa. No caso mais simples, construiu-se um objeto `TDialog` em uma função de resposta de comando da classe da janela-mãe. Executou-se o diálogo - não para eliminá-lo, mas para pôr os mecanismos internos em ação. Ao executar um diálogo, o tornamos visível, exibindo seus controles e habilitando os usuários a entrar com informações nesses controles.

Para finalizar, o primeiro passo na criação de uma caixa de diálogo foi projetar o seu layout utilizando o Resource Workshop (ferramenta disponível junto com o ambiente de programação BC4). Em outros casos, digitou-se os comandos do scritpt de recurso no arquivo de programa “pipdsowl.rc”. Identificou-se os diálogos, inseriu-se diversos objetos de controle - botões, texto estático, caixas de listagem, etc - e, por fim, identificou-se esses subelementos numericamente.

Dando continuidade aos recursos da Plataforma Integrada para Processamento Digital de Sinais, serão discutidos agora os procedimentos rotineiros desempenhados pelo projetista de filtros digitais para realizar a síntese e a quantização dos filtros digitais.

Para realizar a síntese de um filtro digital, o arquivo de coeficientes na forma cascata de blocos de segunda ordem deve estar disponível. Ele é obtido através dos passos discutidos nos parágrafos anteriores. Pode-se realizar a síntese de redes organizada como blocos em cascata e como blocos em paralelo. Antes de realizar a síntese da rede organizadas em blocos em cascata, o usuário opta pelo submódulo denominado “cascade ordenation”, que permite que seja feita a re-ordenação dos blocos, preparando-os para serem sintetizados. Uma caixa de diálogo padrão do Windows aparece e solicita ao usuário que informe o nome de arquivo de coeficientes na forma cascata. Após um nome de arquivo ter sido selecionado, uma caixa de diálogo solicitando que entre com os parâmetros da ordenação fica disponível. Esta caixa de diálogo solicita que o usuário entre com o número de bandas, posteriormente com as freqüências inferior e superior de cada banda, freqüência de amostragem, e a seguir com a norma a ser utilizada, quadrática ou infinita. Por fim uma caixa de diálogo aparece, solicitando a autorização para gerar o filtro ordenado. Se a ordenação for feita com sucesso uma caixa de diálogo solicitando um nome de arquivo para coeficientes re-ordenados aparece.

Como pode ser observado pelos procedimentos anteriormente descritos, o Windows fornece diversos diálogos comuns tais como de abertura e salvamento. Como muitos programas para Windows utilizam esses mesmos diálogos, os usuários experientes podem executar a plataforma mais facilmente, pois esta emprega os diálogos comuns para operações padronizadas.

Todas as classes de diálogo comum utilizadas pela plataforma foram derivadas da classe `TCommomDialog`, que é derivada da classe `TDialog`. A classe `TOpenSaveDialog`, por exemplo, foi derivada da `TCommonDialog`. Construiu-se um diálogo de menu de arquivo do tipo Open..., Save e Save As... utilizando um objeto de uma classe derivada da `TOpenSaveDialog`. A classe `TFileOpenDialog` foi usada para selecionar nomes de arquivos já existentes e `TFileSaveDialog` para entrar com novos nomes de arquivos. Ambos os diálogos permitiram que os usuários da plataforma mudassem para outros drives e diretórios. As classes de diálogo comuns, tais como `TOpenSaveDialog`, incluíram uma subclasse `TData` que armazenaria diversas opções e informações de diálogo. A subclasse `TData` é diferente para cada diálogo comum usado pela plataforma. Na `TOpenSaveDialog`, por exemplo, `TData` definiu ponteiros `char` para `FileName` e para `Filter`, o qual determinou “strings” que especificaram filtros de algarismos-chave tais como *.ftp e *.cab para limitar os tipos de arquivos representativos de filtros digitais exibidos na janela de diálogo.

Para realizar a síntese do filtro digital na forma de blocos de segunda ordem organizados em cascata, o usuário seleciona no menu “Project” a opção “Cascade Synthesis” onde a plataforma disponibilizou uma caixa de diálogo comum, como a que foi descrita nos parágrafos anteriores, para que o usuário escolha o nome de arquivo de coeficientes de função de transferência cascata ordenados que deseja para fazer a síntese. A seguir é mostrada uma caixa de diálogo que permite que o usuário escolha o

tipo de síntese a ser feita pela plataforma. A síntese pode ser feita gerando-se redes ótima (mínimo ruído), rede sem ciclos limite e rede ótima sem ciclos limite. O usuário deve informar logo em seguida, em outra caixa de diálogo, o tipo de norma utilizada para o escalamento da rede, quadrática ou infinita. Caso a síntese seja realizada com sucesso, uma caixa de diálogo é apresentada para que o usuário informe o nome do arquivo que representará o filtro sintetizado em cascata de blocos de segunda ordem. O usuário pode, se assim desejar, continuar realizando novas sínteses, com o mesmo arquivo de coeficientes ordenados, pois o diálogo principal que permite a escolha do tipo de rede cascata é novamente mostrado. Se o usuário já estiver satisfeito, e quiser passar para a quantização do filtro, por exemplo, pode abandonar o diálogo selecionando a opção “Quit dialog” ou clicando o botão Cancel .

O usuário pode, agora, realizar a quantização, para um determinado número de bits, dos filtros que foram antes sintetizados na forma cascata de blocos de segunda ordem. Deve ser lembrado que o usuário não havia levado ainda em consideração o numero finito necessário para representar os coeficientes do filtro no DSP. Para que o usuário realize a quantização do filtro projetado e sintetizado com representação infinita, a plataforma disponibiliza um diálogo que solicita o nome de arquivo contendo a rede sintetizada. Logo em seguida, outro diálogo solicita o número de bits necessários à quantização do filtro. Com a quantização realizada com sucesso, o usuário pode então armazenar em arquivo os coeficientes quantizados, para que possam ser posteriormente transferidos para a memória do DSP.

Neste ponto pode ser destacado que o usuário já pode implementar o filtro sintetizado na forma cascata de blocos de segunda ordem no DSP. De forma similar, ele pode realizar a síntese e a quantização do filtro na forma paralela de blocos de segunda ordem. Para isso, ele deve escolher no menu da plataforma a opção “Parallel synthesis”, e os diálogos adequados lhe vão sendo abertos, de forma idêntica à forma cascata. A operação de quantização dos coeficientes da rede paralela sintetizada também é similar ao caso cascata.

A partir deste ponto será mostrado como a plataforma permite que o usuário, de posse dos arquivos de coeficientes obtidos, por aproximação, síntese e quantização, possa visualizar gráficos característicos do filtro, inclusive já podendo implementar esses filtros na memória do DSP através da interface que a plataforma mantém entre o IBM-PC e o DSP.

O usuário pode visualizar o gráfico da função de transferência do filtro através do menu “Graphics” acionando o submenu “transfer function”. A plataforma mostra, então, uma janela de diálogo que solicita ao usuário que escolha o tipo de configuração de coeficientes que deseja plotar: forma polinomial, cascata ou paralela. Em seguida a plataforma disponibiliza um diálogo que solicita ao usuário a escolha do nome de arquivo que contém a função de transferência a ser plotada. Escolhido o nome de arquivo, o gráfico é então mostrado. A utilização destas ferramentas gráficas permite ao projetista de filtros digitais verificar a adequação do filtro projetado ao gabarito de atenuação desejado.

Os gráficos de redes sintetizadas são visualizados pelo usuário ao selecionar a opção “Synthesized network” no menu “graphics”. O usuário se depara com um diálogo que solicita que escolha o gráfico que deseja visualizar: rede ótima (mínimo ruído), rede sem ciclos limite ou rede ótima sem ciclos limite. Como podem haver dois tipos de conexões, cascata ou paralela, para um mesmo filtro sintetizado, um diálogo solicitando que seja escolhido o tipo de conexão é mostrado. O usuário, ao escolher o tipo de conexão, tem logo a seguir um diálogo que solicita o nome de arquivo contendo a rede a ser plotada. Desta forma o filtro é mostrado graficamente na tela. Convém lembrar que o usuário também pode visualizar o gráfico da rede quantizada com um determinado número de bits. Desta forma o usuário tem uma idéia do filtro projetado com precisão infinita e com número finito de bits. Aqui pode ser destacada uma vantagem do uso do Windows: os vários gráficos podem ser visualizados ao mesmo tempo na tela.

Até aqui foram discutidos apenas alguns conceitos envolvendo diálogos, porém ainda não se descreveu como foi implementada a plataforma, em termos de janelas. É um momento oportuno para dar uma breve explicação sobre como foram implementadas as janelas existentes na plataforma, utilizando os conceitos de Interface de Múltiplos Documentos disponível no Microsoft Windows.

A Interface de Múltiplos Documentos ou MDI (Multiple Document Interface) é uma especificação para aplicativos que lidam com documentos no Microsoft Windows (como é o caso da Plataforma Integrada para Processamento Digital de Sinais). A especificação descreve uma estrutura de janelas e de interfaces que permite ao usuário trabalhar com múltiplos documentos dentro de um único aplicativo (como é o caso das janelas que apresentam os coeficientes do filtro projetado, ou ainda as janelas que mostram gráficos, como é o caso dos gráficos de função de transferência e de redes sintetizadas, etc). Em termos simples, da mesma forma que o Windows mantém múltiplas janelas de documentos dentro de uma única tela, um aplicativo da MDI mantém múltiplas janelas de documentos dentro de uma única área da janela-cliente.

O primeiro aplicativo da MDI para o Windows foi o Microsoft Excel, mas tanto o Gerenciador de Programas quanto o Gerenciador de Arquivos do Windows também são aplicativos da MDI. Embora a especificação MDI esteja disponível desde o Windows 2.0, naquela época os aplicativos eram difíceis de serem codificados e requeriam algumas tarefas bem intrincadas na programação. Entretanto, a partir do Windows 3.0, a maior parte desse trabalho já está pronta.

Como pode ser visto através da plataforma, a janela do aplicativo principal de um programa MDI é convencional - ela tem uma barra de título, um menu, uma moldura para ajustar o tamanho, um ícone de menu de sistema e ícones de maximização e minimização. Entretanto, a área da janela-cliente geralmente é chamada de “espaço de trabalho” e não é usada diretamente para mostrar a saída do programa. Esse espaço de trabalho contém nenhuma ou algumas janelas-filhas, cada uma das quais mostrando um documento (podendo ser os arquivos de coeficientes de filtros, gráficos de função de transferência e de redes sintetizadas, etc).

Essas janelas-filhas, como é o caso da janela de gráfico da função de transferência do filtro, se parecem muito com janelas normais. Elas também possuem uma barra de título, uma moldura, um ícone do menu do sistema, ícones de

maximização e minimização e, em alguns casos, barras de rolagem, como é o caso das janelas-filhas que permitem visualizar as memórias do DSP. Entretanto, como pode ser visto através da plataforma PIPDS, nenhuma das janelas de documento tem um menu. O menu na janela do aplicativo principal aplica-se às janelas de documento.

A qualquer momento, somente uma janela de documento está ativa (indicada por uma barra de título em destaque) e aparece na frente de todas as outras janelas de documentos. Todas as janelas-filhas de documentos são trazidas para dentro da área de trabalho e nunca aparecem fora da janela do aplicativo.

Um nova terminologia é necessária para ver de perto o suporte do Windows à MDI. A janela do aplicativo principal é chamada de "base". Um aplicativo da MDI também cria uma "janela-cliente" com base na classe de janela predefinida `TMDIClient`. Essa janela-cliente cobre a área do cliente da janela de base e é responsável pela maior parte do suporte da MDI.

As janelas de documento são chamadas de "janelas-filhas". As janelas de documento são filhas da janela-cliente, que por sua vez é filha da janela de base. A Figura 4.7. mostra a hierarquia mãe-filha da plataforma PIPDS.

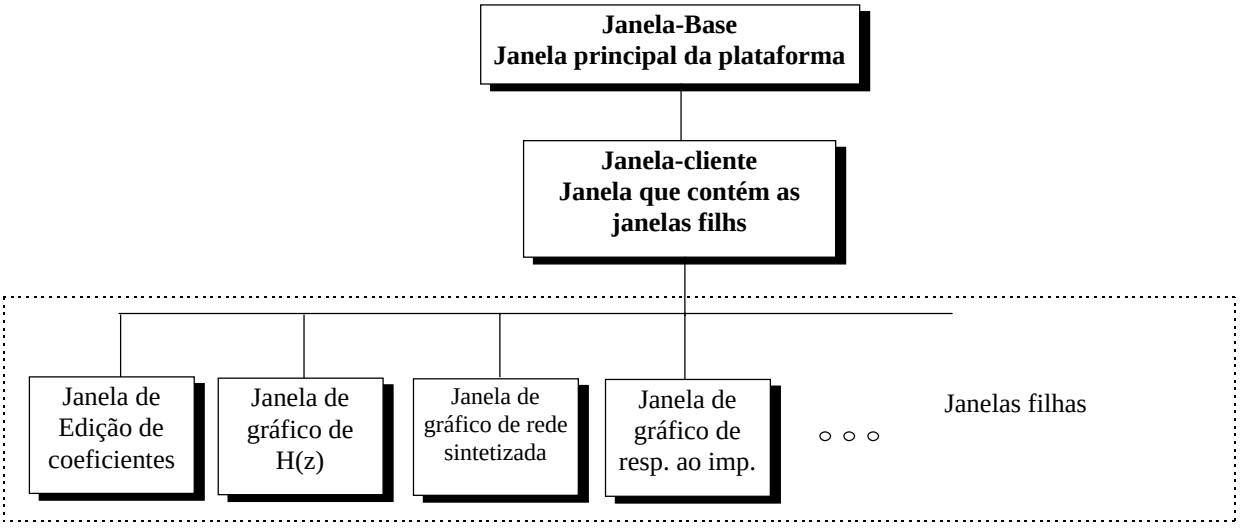


Figura 4.7. A hierarquia mãe-filha da plataforma.

Precisou-se de uma classe de janela (e um procedimento de janela) para a janela de base e para cada tipo de janela-filha suportada pela plataforma.

Como ilustrado pela Figura 4.7, existem três componentes básicos para a plataforma PIPDS:

1. uma janela de base, ou janela-base, que funciona como a janela principal do aplicativo;

2. uma janela-cliente, possuída pela janela de base, que lida com as operações que geralmente se referem ao aplicativo e que criam novas janelas de documento; e

3. várias janelas de documento que exibem informações baseadas em arquivo ou de algum outro tipo (gráficos, por exemplo). Usualmente, cada arquivo aberto, conhecido geralmente como um documento, é designado a uma janela de documento individual.

O serviço primordial da janela de base é possuir uma janela-cliente e fornecer as partes visuais da janela principal do programa. A janela de base mostra a barra de menu do programa (necessária em todos os aplicativos MDI), e ela também possui objetos de janela-filha, tais como barras de ferramentas e linhas de status. Existe somente uma janela de base na plataforma PIPDS.

A janela-cliente da plataforma é vinculada à janela de base, e fornece todas as operações globais - aquelas que se referem ao aplicativo como um todo ou que criam novas janelas-filhas ou abrem arquivos já existentes. Um comando para abrir um diálogo de opções, por exemplo, é tratado pela janela-cliente. Existe somente uma janela-cliente na plataforma PIPDS.

As janelas de documento exibem as informações do documento. Cada janela de documento é uma filha da janela-cliente MDI e responde individualmente aos comandos do menu e a outros eventos, tais como o acionamento do mouse ou operações de teclado. A janela-cliente trata de qualquer evento que não seja processado por uma janela de documento. Esses eventos incluem as operações padrão de janela que podem arrumar em cascata, lado a lado, fechar e rearranjar os ícones das janelas de documento.

A seguir é mostrado como foram colocados em prática os conceitos relacionados acima, utilizando as classes ObjectWindows para criar a plataforma.

A ObjectWindows fornece três classes que foram utilizadas para se desenvolver o aplicativo PIPDS. As classes e seus arquivo-header são:

1. `TDecoratedMDIFrame` - utilizou-se essa classe, declarada em `MDI.H`, para construir objetos de janela de base MDI, que serviram como a janela principal do programa;

2. `TMDIClient` - utilizou-se essa classe, também declarada em `MDI.H`, para construir os objetos de janela-cliente MDI. Um objeto `TDecoratedMDIFrame` possui um objeto da classe `TMDIClient`;

3. `TMDIChild` - utilizou-se essa classe, declarada em `MDICHILD.H`, para construir os objetos de janela-filha de documento. Cada objeto documento é possuído por um objeto da classe `TMDIClient`.

Incluiu-se no projeto da plataforma o arquivo-header de recurso `MDI.RH`, que declara diversas constantes de comando de menu padrão do Windows, tais como `CM_CASCADECHILDREN`. Utilizou-se essas constantes para construir um menu com as opções Cascade, Tile e outros comandos comumente utilizados pelo projetista de filtros na operação da plataforma (esses comandos favorecem o projetista, por exemplo, na hora de visualizar os diversos gráficos de cada filtro projetado, permitindo, por exemplo, que ele possa colocar esses gráficos lado a lado para efetuar comparações).

Não foi necessário implementar esses comandos pois eles são tratados automaticamente pelo objeto `TMDIClient`. A plataforma PIPDS, utiliza em seu arquivo-header `CLIENTE.H`, entre outros, os seguintes arquivos-header:

```
            #include <owl/applicat.h>
            #include <owl/mdi.h>
            #include <mdichild.h>
            #include <mdi.rh>
```

Precisou-se, conforme mostrado abaixo, de uma classe de aplicativo, denominado `TMDIFileApp`, derivada de Tapplication e declarada em `APPLICAT.H`, para implementar o aplicativo PIPDS.

```
//{{TApplication = TMDIFileApp}}
class TMDIFileApp : public TApplication
{
      private:
            bool  HelpState;        // Has the help engine been used.
            bool  ContextHelp;      // SHIFT-F1 state (context
                                    sensitive HELP).
            HCURSOR HelpCursor;     // Context sensitive help cursor.

            void SetupSpeedBar (TDecoratedMDIFrame *frame);
            void AddFiles (TFileList* files);

      public:
            TMDIFileApp ();
            virtual ~TMDIFileApp ();

            void CreateGadgets (TControlBar *cb, bool server = false);

            TEditClient *mdiClient;

            // Public data members used by the print menu commands and
Paint routine in MDIChild.
            TPrinter    *Printer;   // Printer support.
            int         Printing;   // Printing in progress.

      //{{TMDIFileAppVIRTUAL_BEGIN}}
      public:
            virtual void InitMainWindow ();
            virtual void InitInstance ();
            virtual bool CanClose ();
            virtual bool ProcessAppMsg (MSG& msg);
      //{{TMDIFileAppVIRTUAL_END}}

      //{{TMDIFileAppRSP_TBL_BEGIN}}
      protected:
            TToolBox* toolBox ;
            void EvNewView (TView& view);
            void EvCloseView (TView& view);
```

```
            void CmHelpAbout ();
            void CmHelpContents ();
            void CmHelpUsing ();
            void EvDropFiles (TDropInfo drop);
             void EvWinIniChange (char far* section);
      //{{TMDIFileAppRSP_TBL_END}}
      DECLARE_RESPONSE_TABLE(TMDIFileApp);
};    //{{TMDIFileApp}}
```

Como mostrado acima, precisou-se substituir a função `InitMainWindow` para criar a janela principal do programa. Isso foi feito declarando-se um ponteiro privado (que também pode ser público ou protegido) para um objeto `TMDIClient` (neste caso `TEditClient`), que se tornará a janela-cliente do programa. Construiu-se essa janela dentro da função `InitMainWindow`, conforme descrito parcialmente abaixo. Em seguida, construiu-se uma instância da classe `TDecoratedMDIFrame`, para a qual passou-se o título do programa (exibido na barra de título da moldura - `Name`), um identificador de recurso de menu (`MDI_MENU`) e uma referência ao objeto janela-cliente (`*mdiClient`). A classe `TDecoratedMDIFrame` requer um recurso de menu pois o "design" de todos os aplicativos MDI deve ter uma barra de menu.

```
//////////////////////////////////////////////////////
// TMDIFileApp
// =====
// Application intialization.
//
void TMDIFileApp::InitMainWindow ()
{
    if (nCmdShow != SW_HIDE)
            nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ?
            SW_SHOWMAXIMIZED : nCmdShow;

    mdiClient = new TEditClient(this);
    DecoratedMDIFrame* frame = new TDecoratedMDIFrame(Name, MDI_MENU,
      *mdiClient, true, this);

    nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ?
      SW_SHOWMAXIMIZED : nCmdShow;

    //
    // Assign ICON w/ this application.
    //
       frame->SetIcon(this, IDI_MULTIFILE/*IDI_MDIAPPLICATION*/);

    //
    // Menu associated with window and accelerator table associated
with table.
    //
    frame->AssignMenu(MDI_MENU);

    //
    // Associate with the accelerator table.
    //
    frame->Attr.AccelTable = MDI_MENU;
```

```
SetupSpeedBar(frame);

TStatusBar *sb = new TStatusBar(frame, TGadget::Recessed,
  TStatusBar::CapsLock | TStatusBar::NumLock |
  TStatusBar::ScrollLock | TStatusBar::Overtype);

frame->Insert(*sb, TDecoratedFrame::Bottom);

SetMainWindow(frame);

frame->SetMenuDescr(TMenuDescr(MDI_MENU));

//
// Borland Windows custom controls.
//
EnableCtl3d(TRUE) ;
EnableBWCC();
toolBox = new TToolBox(frame, 1);     // Caixa de ferramentas de
                                      apenas uma coluna
toolBox->Insert(*new TButtonGadget(IDB_ASSEMBLAR,
  CM_MENUDEASSEMBLAR));
//toolBox->Insert(*new TButtonGadget(CM_MENUDEBANCO,
  CM_MENUDEBANCO));
toolBox->Insert(*new TButtonGadget(IDB_CONVERTER,
  CM_MENUDECONVERSAO));
toolBox->Insert(*new TButtonGadget(IDB_MOSTRAR,
  CM_MENUDEMOSTRAR));
//toolBox->Insert(*new TSeparatorGadget(3));
toolBox->Insert(*new TButtonGadget(IDB_PREENCHER,
  CM_MENUDEPREENCHER));
toolBox->Insert(*new TButtonGadget(IDB_INICIAR,
  CM_MENUDEEXECUTAR));
toolBox->Insert(*new TButtonGadget(IDB_PARAR, CM_MENUDEPARAR));
toolBox->Insert(*new TButtonGadget(IDB_ENTRARPELAPORTA,
  CM_MENUDEENTRARPELAPORTA));
//toolBox->Insert(*new TSeparatorGadget(3));
//toolBox->Insert(*new TButtonGadget(CM_MENUDELOGAR,
  CM_MENUDELOGAR));
//toolBox->Insert(*new TButtonGadget(CM_MENUDEMODIFICAR,
  CM_MENUDEMODIFICAR));
toolBox->Insert(*new TButtonGadget(IDB_SAIRPELAPORTA,
  CM_MENUDESAIRPELAPORTA));
//toolBox->Insert(*new TSeparatorGadget(3));
//toolBox->Insert(*new TButtonGadget(CM_MENUDEPAGINA,
  CM_MENUDEPAGINA));
//toolBox->Insert(*new TSeparatorGadget(3));
//toolBox->Insert(*new TButtonGadget(CM_MENUDELERARQUIVO,
  CM_MENUDELERARQUIVO));
//toolBox->Insert(*new TSeparatorGadget(3));
toolBox->Insert(*new TButtonGadget(IDB_SUBSTITUIR,
  CM_MENUDESUBSTITUIR));
toolBox->Insert(*new TButtonGadget(IDB_DESASSEMBLAR,
  CM_MENUDEDESASSEMBLAR));
//toolBox->Insert(*new TButtonGadget(CM_MENUDEVISUALIZAR,
  CM_MENUDEVISUALIZAR));
```

```
    toolBox->Insert(*new  TButtonGadget(IDB_SINTAXE,
      CM_MENUDESINTAXE));
    frame->Insert(*toolBox, TDecoratedFrame::Left) ;
    toolBox->Attr.Style |= WS_CLIPSIBLINGS;
    toolBox->Attr.Id = IDW_TOOLBAR;
    // Add fly-over help hints.
    toolBox->SetHintMode(TGadgetWindow::EnterHints);
}
```

Construir um objeto de `TDecoratedMDIFrame` inicia uma procura por um menu de janela. Se o objeto encontra esse menu, ele anexa automaticamente os títulos das janelas-filhas de documento na parte inferior do menu. Quando se fecha uma janela de documento, o nome dela é removido automaticamente do menu. É o que pode ser visto quando o projetista de filtros visualiza um gráfico ou edita um arquivo contendo coeficientes. Com a ajuda da classe `TDecoratedMDIFrame`, pode-se decorar as janelas de moldura MDI. A listagem encontrada no arquivo `PIPDSOWL.CPP` demonstra como foi utilizada a classe para criar um aplicativo MDI completo com uma barra de ferramentas e uma linha de status. Pode-se observar na Figura 4.8 abaixo que o aplicativo tem muitas das características de um aplicativo Windows.

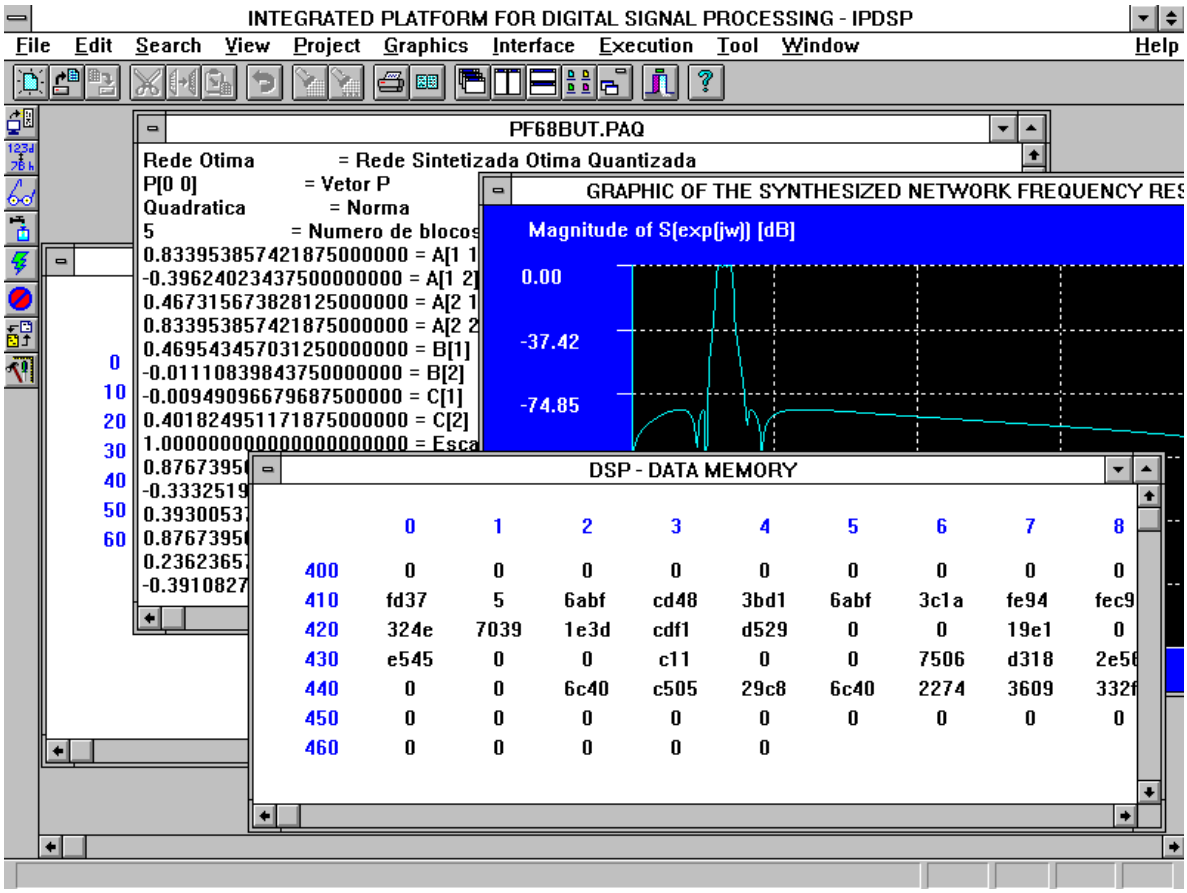


Figura 4.8. Características de moldura na plataforma.

Identificou-se cada ícone e comando de menu com valores numéricos ou “string” definidos no arquivo-header de recurso `PIPDSOWL.RH`. Os ícones da barra de ferramentas enviam as mesmas mensagens de comando dos menus - eles são apenas

meios visualmente diferentes e, normalmente mais convenientes, dos usuários selecionarem os comandos de um programa. Na listagem `PIPDSOWL.RH`, também foram definidos diversos identificadores, alguns dos quais (tal como `CM_FILESAVE`) estão duplicados no ObjectWindows. Deve ser observado que ao utilizarmos os identificadores padrão, automaticamente são exibidas sugestões na barra de status, quando esses comandos são selecionados.

Incluiu-se o arquivo-header `DECMDIFR.H`, que declara a classe `TDecoratedMDIFrame`. A classe é derivada da classe `TMDIFrame` e da classe `TDecoratedFrame`, combinando efetivamente os elementos encapsulados das janelas de moldura MDI com molduras decoradas de janela-única que podem possuir barras de ferramentas e outros dispositivos.

Em seguida, construiu-se uma barra de ferramentas como um objeto da classe `TControlBar`, para o qual passou-se o endereço do objeto janela de base (`frame`). Inseriu-se vários dispositivos de botão de ícone, conforme mostrado na barra de ferramentas, especificando um identificador de recurso de ícone e o comando de menu que deverá ser avisado quando os usuários selecionarem o botão. Utilizou-se a classe `TSeparatorGadget` para criar espaço entre os grupos de botões. Chamou-se a função `SetHintMode`, conforme mostrado, para habilitar as sugestões on-line para os botões tocados (mas não necessariamente clicados) pelo ponteiro do mouse. Finalmente, inseriu-se a barra de ferramentas no objeto janela de moldura MDI decorada. Criou-se uma linha de status de modo similar, que também foi inserida na moldura.

Cada classe MDI possui diversas funções-membro que são chamadas para realizar várias tarefas. A classe `TDecoratedMDIFrame` foi usada apenas para construir a janela principal do programa. Entretanto, a classe `TEditClient`, derivada de `TMDIClient`, possui diversas funções que precisaram ser chamadas, conforme pode ser visto na declaração da classe que se encontra no arquivo `CLIENTE.H`.

Em um procedimento comumente utilizado pelo usuário, como é o caso de se realizar um “zoom” de um gráfico selecionado, é necessário obter o endereço de uma janela-filha de documento atualmente ativa. Para isso chamou-se a função `GetActiveMDIChild`, conforme mostrado abaixo.

```
//
// Rotina ZoomHz()
//

void TEditClient::ZoomHz()
{

      double wminnovo, wmaxnovo, valminnovo, valmaxnovo ;

      TDrawChild* cp = (TDrawChild *)GetActiveMDIChild() ;

      if (!cp)
            return ;

      else
      {
```

```
            strcpy(dialogozoomhz.WminHzOn, "0") ;
            strcpy(dialogozoomhz.WmaxHzOn, "3.14") ;
            strcpy(dialogozoomhz.ValminHzOn, "-300") ;
            //strcpy(dialogozoom.ValmaxOn, "0") ;

            TOptionDialog* DialogoZoomHz = new TOptionDialog(this,
&dialogozoomhz) ;

            if (DialogoZoomHz->Execute() == IDOK)
            {
                  wminnovo = atof(dialogozoomhz.WminHzOn) ;
                  wmaxnovo = atof(dialogozoomhz.WmaxHzOn) ;
                  valminnovo = atof(dialogozoomhz.ValminHzOn) ;
                  //valmaxnovo = atof(dialogozoom.ValmaxOn) ;
                  valmaxnovo = 0 ;
            }

            cp->Grafico(wminnovo, wmaxnovo, valminnovo, valmaxnovo) ;

            return ;

      } // fim do if

}
```

Na opção que permite visualizar um gráfico, durante sua programação, houve um momento que foi necessário chamar a função `Create`, herdada da classe `Twindow`, para criar a janela para visualização do gráfico. Como exemplo, pode-se citar o caso da criação da janela-filha para visualizar o gráfico de uma rede sintetizada.

```
            TRedeChild *cp;

            cp = new TRedeChild(valmax, valmin, mod_h, phase_h,
options.ModuloOn, options.FaseOn, *this, npontos);

            // Associate ICON w/ this child window.
            cp->SetIcon(GetApplication(), IDI_DOC);
            cp->Create();
            cp->Grafico(0, 3.14, valmin, 0);
```

Logo em seguinda, é mostrado um trecho de definição de classe para esta mesma janela-filha:

```
class TRedeChild: public TMDIChild
{

public:
      OptionBuffer options ;
      TRedeChild::TRedeChild(double e, double f, double *Modhrede,
double *Phase_hrede, WORD ModOn, WORD FasOn, TMDIClient& parent, int
numerodepontos) ;
      void SetMaxCoordinates() ;
      double *mod_h ;
      double *phase_h ;
```

```
      double valmax, valmin ;
      int graficomodulo, graficofase ;
      void Grafico(double wminnovo, double wmaxnovo, double valminnovo,
double valmaxnovo) ;
      //void Grafico() ;
      int xMax, yMax;    // Coordenadas máximas para a área cliente
      int xMin, yMin ;
      int xBase, yBase;  // Base de coordenadas de X e Y para o gráfico
      int Npont, Indice, Indice1, Cont ;
      double ValMax, ValMin, Modulo, ReNum, ImNum, ReDen, ImDen, Re,
Img, wmax, wmin, w ;
      int yorigem ;
      double valx, valy ;
      int npontos ;
      int numerodepontos, contw ;
      double passo ;

protected:
      void InitDrawChild();
      void Paint(TDC &dc, BOOL erase, TRect& rect);
      void EvSize(UINT sizeType, TSize& size) ;

private:
      string fileName;  // Nome do arquivo como um objeto string do C++
      string* filename ;
      DECLARE_RESPONSE_TABLE(TRedeChild) ;
};
```

Folheando-se as listagens que compõem a programação da plataforma, pode-se observar os passos básicos requeridos pelo aplicativo MDI para criação de janelas-filhas. Primeiro, derivou-se as classes `TDrawChild`, `TRedeChild`, `TRedeRimp`, `TJanelaMostrarMemoria` e `TJanelaDesassemblarMemoria`, a partir da classe `TMDIChild`. Os objetos das suas classes se tornaram então as janelas-filhas de documento do programa. Em seguida, foram inseridas todas as funções-membro de dados que foram necessárias para abrir, salvar, editar e exibir os dados de um documento. Para cada tipo de documento tratado pela plataforma derivou-se uma classe `TMDIChild`.

Como pode ser observado nas listagens do arquivo `CLIENTE.H`, as classes referenciadas acima possuem um construtor, que constroi uma nova janela de documento. Um dos parâmetros requeridos foi uma referência ao pai do objeto `TMDIClient` da janela-filha. Passou-se esse parâmetro para o construtor da classe de base `TMDIClient`.

A algumas classes foram acrescentadas funções que denotaram as tarefas que as janelas-filhas deveriam realizar. Armazenou-se, também, um nome de arquivo em um objeto da classe “string”.

As janelas-filhas MDI da plataforma são mais independentes do que outras janelas-filhas como botões de acionamento e outros controles. O projetista de filtro pode

notar que uma janela-filha MDI é uma janela sobreposta que está restrita às bordas da sua janela-mãe, mas ela pode ser redimensionada, reduzida até um ícone, ou expandida para preencher a janela-mãe. As janelas filhas MDI não podem ser movidas para fora das bordas de sua janela-mãe. As janelas-filhas MDI também não podem ter menus, mas podem ser controladas pelos comandos da barra de menu da janela de moldura.

Em algumas situações, como no momento de editar um arquivo de coeficientes, por exemplo, os títulos de janela-filha MDI foram constituídos pelo nome de arquivo dos documentos editados. Na plataforma, também foram nomeadas algumas janelas-filhas com algo que pudesse representar a sua função. Isto foi implementado da seguinte maneira:

```
//
// Construtor
//

TRedeChild::TRedeChild(double e, double f, double *Modhrede, double
*Phase_hrede, WORD ModOn, WORD FasOn, TMDIClient& parent, int
numerodepontos ) : TMDIChild(parent, "GRAPHIC OF THE SYNTHESIZED
NETWORK FREQUENCY RESPONSE")
{

      InitDrawChild();

      mod_h = Modhrede ;
      phase_h = Phase_hrede ;

      options.ModuloOn = ModOn ;
      options.FaseOn = FasOn ;

      npontos = numerodepontos ;

} // fim da rotina do Construtor
```

Após programar as classes derivadas da `TMDIChild`, derivou-se uma classe de janela-cliente a partir da `TMDIClient`. Na listagem `CLIENTE.H` pode ser observado que o aplicativo necessita de somente uma classe derivada, denominada `TEditClient`. Isso foi feito conforme exposto logo abaixo:

```
      //{{TMDIClient = TEditClient}}
      class TEditClient : public TMDIClient
      {

      public:
             int ChildCount; // Number of child window created.

             TEditClient(TModule* module = 0);
             virtual ~TEditClient ();

             void OpenFile (const char *fileName = 0);

             OptionBuffer options ;
             GrafRedeRimp grafrederimp ;
             OpcaoProjAproxDigitalMenudeTipos
opcaoprojaproxdigitalmenudetipos ;
```

```
            OpcaoProjAproxMenudeTiposSinteseCascata
            opcaoprojaproxmenudetipossintesecascata ;

            .
            .
            .
      }
```

A classe da janela-cliente tem um construtor. A janela de base MDI possui uma janela-cliente, e a janela-cliente não precisa se referir à sua janela-mãe, de modo que o construtor não precisa de parâmetros. Isto foi implementado do seguinte modo:

```
      TEditClient() : TMDIClient(){}
```

Uma outra janela filha importante a ser destacada é a que permite que o usuário verifique o resultado da implementação no DSP do filtro projetado. Isto é feito através da visualização da resposta ao impulso do filtro. Porém, antes é importante discutir como o usuário se relaciona com o DSP começando com a transferência de coeficientes para sua memória de dados.

O usuário ao decidir transferir os coeficientes para a memória de dados do DSP tem disponibilizado pela plataforma um diálogo que solicita que ele informe a freqüência para programação dos conversores A/D e D/A. Logo a seguir, a plataforma pede que seja informado o tipo de conexão de blocos de segunda ordem e posteriormente o correspondente arquivo de coeficientes do filtro já quantizado.

Para um melhor entendimento de como esses parâmetros são transferidos para a memória do DSP é importante darmos uma pequena demonstração de como foram programados os conversores do DSP.

Após a freqüência de amostragem ser solicitada ao usuário, sua especificação é usada para programar o timer (constituído de dois contadores de 8 bits conectados em cascata) dos conversores A/D e D/A da placa baseada de DSP. Cada contador de 8 bits está associado a um divisor. Portanto, dependendo da freqüência de entrada deste contador e de seu divisor programado, a sua freqüência de saída será uma fração de sua freqüência de entrada.

Portanto para o DSP, que usa como freqüência de entrada o valor de 4 Mz, sua freqüência de saída é calculada da seguinte forma:

$$\text{Freqüência de saída do clock} = 4/(div1*div2)$$

Se desejarmos uma freqüência de saída de 100 kHz, por exemplo, o produto div1*div2 será 40, logo uma combinação para os divisores poderia ser div1 = 40 e div2 = 1.

Para programarmos o timer, deve ser enviado para a porta 0 do Timer do DSP o valor denominado `TheValue` (variável fictícia), onde o valor de `TheValue` é dado por:

TheValue = time1time2

onde

time1 = 255 - div1

e

time2 = 255 - div2

Para o caso citado acima, usando-se div1 = 40 e div2 = 1 obtém-se time1 = 215 ou 0D7h e time2 = 254 ou 0feh (aqui os valores terminados por "h" estão representados na base hexadecimal).

Portanto

TheValue = 0FED7h.

Para que este parâmetro seja transferido para a memória de dados do DSP é necessário ao IBM-PC executar algumas rotinas de forma conveniente. Entre algumas delas podemos citar a função `sendio` que possibilita que um vetor, ou elemento de um vetor (por exemplo, a freqüência de amostragem) seja transferido para a memória de dados ou de programa do DSP.

A seguir é mostrada a rotina `sendio`:

```
void TEditClient::sendio(unsigned int * xs, unsigned int len, unsigned
int adr320, unsigned int iobase, unsigned int bankpd, unsigned int
bank12pc)
{
      unsigned int xsSEG, xsOFF;

      xsSEG = FP_SEG(xs);
      xsOFF = FP_OFF(xs);
      setupadr(adr320, bankpd, bank12pc, iobase);
      asm
      {
            push ds ;
            push si ;
            mov dx, iobase ;
            mov ax, xsSEG ;
            mov ds, ax ;
            mov si, xsOFF ;
            mov cx, len ;
            cld ;
            rep outsw ;
            pop si ;
            pop ds ;
      }
} // fim da rotina sendio()
```

Caso o DSP esteja executando um programa e for feita uma tentativa de transferência de algum tipo de parâmetro para sua memória, a plataforma se encarrega de interromper a operação do DSP através da rotina denominada `hlt320`. Trecho desta rotina se encontra listado logo abaixo e pode ser encontrado no arquivo `TRANBUFF.CPP`:

```
void TEditClient::hlt320(unsigned int iobase)
{
      inportb(iobase + 7) ;
} // fim da rotina hlt320()
```

Utilizando-se a constante 300h como endereço base de entrada e saída no IBM-PC, a instrução `inportb` enviará um byte para o endereço 307h e o DSP será interrompido.

Procedimentos similares são realizados para a transferência de coeficientes para a memória de dados do DSP.

Antes de mostrar como o usuário visualiza a resposta ao impulso de um filtro será mostrado como o programa ".asm" correspondente a um filtro é compilado.

Um programa é compilado, ou seja seu código objeto é gerado e transferido para a memória de programa do DSP, quando o usuário escolhe a opção "Assembly" no menu "Execution". Ao fazer essa escolha, a plataforma mostra uma caixa de diálogo onde o usuário informa o nome do arquivo ".asm" que será compilado. Caso o programa seja compilado com sucesso, a caixa de diálogo correspondente informará ao usuário. Caso contrário, o usuário terá uma lista de erros encontrados onde ele, através do editor da plataforma poderá editar novamente o programa. Destaque-se, aqui, a total integração da plataforma, onde o usuário está interagindo em ambiente baseado no IBM-PC e ao mesmo tempo com um processador dedicado, transferindo programas para sua memória e/ou editando-os. Uma vez editado/depurado o programa o usuário poderá novamente compilá-lo para a memória do DSP e posteriormente executá-lo. Para executar o programa a plataforma possui a opção "Go" do menu "Execution".

Estas tarefas estão implícitas quando o usuário planeja verificar a resposta ao impulso do filtro projetado. Antes de rodar a resposta ao impulso, a plataforma informa que o usuário deve enviar os coeficientes do filtro quantizado para a memória de dados do DSP e posteriormente informar o valor da amplitude do sinal de entrada do filtro. Notemos que o usuário ainda não está executando o filtro em tempo real. Assim, ele preenche um "buffer" de saída uma única vez, e o deixa disponível na memória de dados do DSP. É necessário, então, executar uma rotina de Transformada Rápida de Fourier, no ambiente do IBM-PC, para converter o resultado obtido no domínio do tempo para o domínio da freqüência. O "buffer" usado contém 2048 amostras, e seu preenchimento é feito de forma sequencial, através de um programa ".asm" semelhante ao desenvolvido para processar o filtro em tempo real. Uma vez obtida a resposta em freqüência do filtro implementado no DSP, o usuário pode visualizar o gráfico do módulo ou da fase através de um diálogo para este fim. O aspecto da janela que permite visualizar este tipo de gráfico é idêntico aos das janelas-filhas já discutidos nos parágrafos anteriores. O usuário pode agora comparar o gráfico com o gabarito de

atenuação especificado para o filtro, podendo, para isso, dispor as janelas lado a lado, realizar o zoom de determinada região do gráfico, etc.

Antes de passar às ferramentas de depuração de programas e visualização do conteúdo das memórias do DSP, vamos discutir a implementação do Analisador de Sinais, uma ferramenta associada à Plataforma Integrada para Processamento Digital de Sinais, que permite ao usuário acompanhar em tempo real a operação do filtro no DSP. Este analisador roda em tempo real graças ao temporizador do Windows.

O usuário pode acessar o Analisador de Sinais através da opção "Signal Analyser" do menu Execution. Uma vez acessado o Analisador, uma outra janela, com estrutura idêntica à da plataforma PIPDS é mostrada. Nesta janela o usuário pode selecionar o tipo de ferramenta que gostaria de visualizar. Estão disponíveis gráficos temporais dos sinais de entrada e saída (isoladamente ou em conjunto), representados por um Osciloscópio, bem como os gráficos espectrais dos mesmos sinais, também isoladamente ou em conjunto, representado por um Analisador de Espectro.

O Analisador de Sinais, programado para o ambiente Windows através da biblioteca de classes OWL, utiliza uma estrutura de janelas idêntica à que foi utilizada pela plataforma PIPDS.

Derivou-se uma classe a partir da classe `Tapplication` denominada `TAnaSinalApp`, para implementar o aplicativo. Em seguida, derivou-se uma classe denominada `TAnaSinalCliente`, a partir da classe `TMDIClient`, para implementar a janela cliente que comportará as janelas filhas que permitirão visualizar o Osciloscópio e o Analisador de espectro. Por fim, foi implementada a temporização no Analisador de Sinais, permitindo um acompanhamento em tempo real dos sinais envolvidos no filtro implementado no DSP.

O Analisador de Sinais é uma ferramenta controlada pelo temporizador do Windows. A programação desse timer é feita a partir de sua inicialização na função `SetupWindow`, função membro da classe `TAnaSinalCliente`. Nessa implementação, a função `SetupWindow` inicializou o timer com a seguinte declaração:

```
//
// Setup the main window, and try to allocate its timer
//
void TAnaSinalCliente::SetupWindow()
{
      TMDIClient::SetupWindow();

      int result = IDRETRY;
      while (SetTimer(0, 1000, 0) == 0 && result == IDRETRY)
            result = MessageBox("Cannot create Timer", "Signal
            Analyser", MB_RETRYCANCEL);
      if (result == IDCANCEL)
            PostQuitMessage(0);
}
```

O primeiro elemento de `SetTimer()` é representado pelo parâmetro `TIMER_ID`. Este parâmetro é apenas um identificador para que o programa possa se referir a esse mesmo timer posteriormente. No Analisador de Sinais, o `TIMER_ID` usado é igual a 0, porém poderia assumir um outro inteiro qualquer. O segundo parâmetro de `SetTimer()` é o parâmetro `TIMER_RATE`, padronizado pelo Windows em milisegundos, indicando, neste caso, 1000 milisegundos, ou seja, um tempo de "refresh" de tela de aproximadamente 1 segundo. A taxa máxima é de 65.535 milisegundos - cerca de um minuto.

Depois que o programa chama a função `SetTimer()`, o Windows começa a emitir mensagens `WM_TIMER` na freqüência que foi especificada, ou seja, em intervalos de 1 segundo. O Analisador de Sinais intercepta essas mensagens com uma função de resposta denominada `EvTimer()`, declarado na classe `TAnaSinalCliente`. A classe também insere a macro message-cracking `EV_WM_TIMER` em uma tabela de respostas. Para interromper o timer, o destruidor de `TAnaSinalCliente` chama `KillTimer()`, passando o parâmetro `TIMER_ID` como argumento identificador do timer que estava sendo usado.

A função `EvTimer()` é chamada toda vez que a janela recebe uma mensagem `WM_TIMER`. Como isso acontece mesmo quando a janela de outro aplicativo está ativa, o Analisador de Sinais continua a rodar em background independentemente do que esteja acontecendo. Algumas operações críticas, entretanto, como acesso a disco e a tarefa de arrastar uma janela, interrompem temporariamente os processos em background implementados com o uso desses timers.

Todas as amostras dos sinais de entrada e de saída do filtro em execução no DSP devem ser armazenados em dois "buffers" criados em sua memória de dados (ver Apêndice A), tornando-se assim acessíveis ao Analisador de Sinais. O comprimento de tais "buffers" foi definido como 2048 amostras, suficientes para uma boa precisão na amostragem das FFT's /Ant93/, quando da geração dos gráficos espectrais. Como não há sincronismo entre os dois processadores (e nem poderia haver, dadas suas tarefas), tais "buffers" são preenchidos pelo DSP, e lidos pelo computador hospedeiro, em forma de fila circular, que contém um apontador para a posição da amostra mais recente, visando garantir a informação correta do sequenciamento das amostras. Porém, é necessário destacar que isso traz alguns problemas para o usuário. Como a leitura e processamento dos "buffers" é muito mais lento que o cálculo, pelo DSP, das 2048 amostras do "buffer" de saída, pois o aumento da freqüência de amostragem pode levar à perda do controle do apontador da amostra mais recente dentro dos "buffers", durante sua leitura, havendo interpretação incorreta dos valores lidos. Assim, o usuário deve estar ciente de que para altas freqüências de amostragem não é viável o uso do Analisador de Sinais.

O tempo de "refresh" de aproximadamente um segundo possibilita que seja feito o processamento em tempo hábil das 2048 amostras dos buffers de entrada e saída, que são necessárias para a realização da FFT para o Analisador de Espectro.

A cada intervalo um procedimento similar ao de envio de coeficientes é executado pelo IBM-PC, e um vetor de um determinado tamanho é transferido da

memória de dados do DSP para o PC. Esta programação está representada parcialmente pela função `recebebuffer` descrita abaixo:

```
//
// Rotina recebebuffer()
//

void TAnalisador::recebebuffer(unsigned int * xs, unsigned int len,
unsigned int adr320, unsigned int iobase, unsigned int bankpd,
unsigned int bank12pc)
{
      unsigned int xsSEG, xsOFF;

      xsSEG = FP_SEG(xs);
      xsOFF = FP_OFF(xs);

      setupendereco(adr320, bankpd, bank12pc, iobase);

      asm
      {
            push es ;
            push di ;
            mov dx, iobase ;
            mov ax, xsSEG ;
            mov es, ax ;
            mov di, xsOFF ;
            mov cx, len ;
            cld ;
            rep insw ;
            pop di ;
            pop es ;
      }
} // fim da rotina recvio()
```

Neste ponto, serão destacadas as ferramentas disponíveis ao usuário para que possa depurar/corrigir programas compilados para o DSP. Uma ferramenta bastante usual é a de edição de arquivos ASCII, incluindo aí os programas fonte para o DSP. Ao compilar um código fonte, através de ferramenta disponível para este fim, pode haver necessidade de corrigir eventuais erros decorrentes da edição do código original. Para isso, a plataforma indica, logo após a compilação, através de janela programada para este fim, os erros encontrados. O usuário pode editar novamente o programa fonte através do editor da plataforma, e corrigir os erros.

Uma outra situação é aquela em que o usuário precisa visualizar o código objeto transferido para a memória de programa do DSP, ou mesmo os coeficientes, número de blocos e freqüência de amostragem para programação dos conversores, que estão armazenados na memória de dados. Uma janela-filha está disponível para este fim. O usuário, desejando visualizar o conteúdo da memória de programa ou de dados, deve acessar o botão denominado “Display memory” disposto na lateral esquerda da plataforma e uma janela de diálogo estará disponível solicitando ao usuário que informe qual a memória do DSP ele deseja visualizar, bem como o intervalo de endereços na referida memória. Convém observar que caso o usuário deseje visualizar um conteúdo de memória superior à capacidade de informação da janela-filha (mesmo maximizando-a) um recurso de rolamento do conteúdo da janela está implementado. O usuário, através de barras dispostas do lado direito e inferior da janela, pode acessar outros endereços que antes não estavam visíveis, limitado, obviamente, ao intervalo originalmente

selecionado. As barras de rolagem para a janela-filha foram implementadas através da função `InitDrawChild` declarada na classe `TJanelaMostrarMemoria` como descrito abaixo:

```
//
// Rotina InitDrawChild()
//

void TJanelaMostrarMemoria::InitDrawChild()
{
      //Attr.Style = Attr.Style | (WS_VSCROLL | WS_HSCROLL);
      //Scroller = new TScroller(this, 1, 1, 200, 200);
      Attr.X = GetSystemMetrics(SM_CXSCREEN) / 8;
      Attr.Y = GetSystemMetrics(SM_CYSCREEN) / 8;
      Attr.H = Attr.Y * 6;
      Attr.W = Attr.X * 6;
} // fim da rotina InitDrawChild()
```

Uma outra tarefa comumente desempenhada pelo projetista de filtros digitais diz respeito à visualização do código fonte armazenado na memória de programa do DSP. Para isso, o usuário conta com uma ferramenta que converte o conteúdo dos endereços da memória de programa novamente em programa fonte. O usuário deve selecionar a opção denominada “Disassembly” disponível na barra de ferramentas disposta do lado esquerdo da plataforma. Logo a seguir é solicitado que o usuário informe os endereços inicial e final, e por fim a plataforma mostra o conteúdo convertido através de uma janela-filha programada para este fim. Deve ser notado que a janela foi programada com o mesmo recurso de rolamento disponível para a janela de visualização do conteúdo das memórias, descrita anteriormente, porém a função utilizada para este fim foi declarada na classe `TJanelaDesassemblarMemoria` derivada da classe `TMDIChild.`

Uma outra ferramenta de grande valor para o projetista de filtros é a execução de programas contidos na memória do DSP até um endereço previamente informado. Isso é possível através do botão denominado “Go-breakpoint” onde o usuário visualiza uma janela de diálogo onde é solicitado que informe o endereço para o ponto de parada. Após informar o endereço do breakpoint, a plataforma mostra uma janela consistindo dos valores contidos em vários registradores internos do DSP.

Uma técnica semelhante à programação da plataforma para que disponibilizasse a execução do programa no DSP com “breakpoints” foi usada para que seja possível ao usuário medir o tempo de processamento de um determinado programa armazenado na memória do DSP. A plataforma, de posse dos endereços fornecidos pelo usuário, insere um “breakpoint” nestes dois pontos, medindo-se o tempo gasto desde o início do programa até o primeiro breakpoint e o tempo gasto ao executar novamente o programa até encontrar o segundo breakpoint. Ao computar a diferença entre os dois intervalos de tempo, tem-se o tempo gasto no intervalo entre o primeiro endereço e o segundo. Esse cálculo é feito levando-se em conta a arquitetura Harvard do processador do DSP, computando o número de ciclos gastos para execução do número de instruções daquele intervalo. Esses procedimentos estão descritos em detalhes através de comentários no programa “DEBUGA.CPP”.

Dessa forma, a plataforma PIPDS, assim como o Analisador de Sinais a ela associado, foi inteiramente descrita, não só do ponto de vista das ações que ela permite

ao usuário como do ponto de vista da sua programação, em termos dos objetos derivados e suas características principais. Para completar tal descrição, o Capítulo 5 mostra, passo a passo, um exemplo de utilização das facilidades disponíveis ao usuário. É importante observar, a esta altura, que seria muito mais difícil implementar todas as facilidades colocadas à disposição do usuário, se não fosse usada uma interface gráfica de usuário, dentro do padrão Windows.

CAPÍTULO 5

Exemplo de Projeto e Implementação de Um Filtro Digital Utilizando-se a Plataforma Integrada para Processamento Digital de Sinais

Neste capítulo, serão apresentados os principais passos, e os correspondentes resultados, referentes à implementação em DSP de um filtro digital realizado via rede descrita no espaço de estados utilizando-se a estrutura de segunda ordem da Figura 2.7 como bloco básico, para ilustrar a utilização da plataforma PIPDS.

O filtro digital dessa experiência é um filtro passa-faixa Elíptico, com as seguintes especificações:

Especificação	Valores
Freqüência de amostragem - fs	10000
Freqüência limite da banda de rejeição inferior - f1	500
Freqüência inferior da banda de passagem - f2	600
Freqüência superior da banda de passagem - f3	800
Freqüência limite da banda de rejeição superior - f4	900
Ripple máximo na banda passante - Ap	1.0
Ordem do filtro - n	10

O filtro IIR assim definido tem como função de transferência

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{N} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}} = \frac{b_0 + b_1 z^{-1} + ... + b_N z^{-N}}{1 + a_1 z^{-1} + ... + a_N z^{-N}}$$

(5.1)

que equivale a

$$H(z) = \prod_{i=1}^{N/2} \frac{a_2 z^2 + a_1 z + a_0}{b_2 z^2 + b_1 z + b_0}$$

(5.2)

ou ainda a

$$H(z) = D + \sum_{i=1}^{N/2} \frac{a_2 z^2 + a_1 z + a_0}{b_2 z^2 + b_1 z + b_0}$$

(5.3)

considerando N par, sem perda de generalidade.

Os coeficientes da função de transferência do filtro aqui implementado foram obtidos através do sub-módulo "Aproximation" da Plataforma Integrada para Processamento Digital de Sinais - PIPDS, utilizando-se a aproximação Elíptica. Então, é gerado um arquivo contendo os coeficientes de H(z) na forma polinomial extendida, na forma cascata de blocos quadráticos ou na forma paralela de blocos quadráticos, conforme seja usada a representação das equações (5.1), (5.2) ou (5.3), respectivamente.

Uma das quatro redes variáveis de estado de segunda ordem discutidas no Capítulo 2 pode, então, ser utilizada para realizar funções de transferência na forma cascata ou na forma paralela de blocos de segunda ordem, a partir dos arquivos descritivos das equações (5.2) ou (5.3). Então os sub-módulos denominados "Cascade Ordenation", "Cascade Synthesis" e "Cascade Quantization", podem ser utilizados para a obtenção dos parâmetros da rede (matriz **A**, vetores **b** e **c**, além do escalar d, para cada seção) sintetizada na forma de blocos de segunda ordem organizados em cascata, usando o escalamento em norma quadrática e posteriormente quantizando tais coeficientes para 16 bits.

Já os sub-módulos denominados "Parallel Synthesis" e "Parallel Quantization" podem ser utilizados para a obtenção dos parâmetros da rede sintetizada na forma de blocos de segunda ordem organizados em paralelo. Antes de sintetizar a rede, nesse caso, poderia ser usada uma opção de gerar a função de transferência em blocos de segunda ordem organizados em paralelo, conforme a equação (5.3), armazenando-se o novo conjunto de coeficientes de H(z) em um novo arquivo, a partir do qual a síntese da rede e a quantização dos coeficientes resultantes seriam realizadas.

Para o exemplo demonstrativo, é selecionada a rede na forma paralela de seções ótimas de segunda ordem, para a síntese do filtro. Para implementá-lo no DSP TMS320C25, foi desenvolvido um programa para o cálculo de y(n) a partir da entrada u(n). Para a execução do filtro (na forma paralela), o programa assembly pode ser sumarizado no seguinte algoritmo, ativado por uma interrupção sincronizada com o período de amostragem (mais uma vez, apenas a seção ótima é considerada):

1. lê uma amostra do sinal de entrada u(k);

2. calcula a saída de cada bloco de segunda ordem, dada por

$$y_n(k) = c_{n1}X_{n1}(k) + c_{n2}X_{n2}(k);$$

3. calcula os novos estados, para cada bloco de segunda ordem, dados por

$$X_{n1}(k + 1) = a_{n11}X_{n1}(k) + a_{n12}X_{n2}(k) + b_{n1}u(k)$$
$$X_{n2}(k + 1) = a_{n22}X_{n2}(k) + a_{n21}X_{n1}(k) + b_{n2}u(k) ;$$

4. Calcula a saída local da rede paralela, dada por

$$y(k) = y_1(k) + y_2(k) + ... + y_N(k) + du(k);$$

5. envia para o periférico de saída a resposta calculada do filtro, y(k);

6. aguarda nova interrupção.

Um fluxograma baseado nesse algoritmo, mostrado na Figura 5.1, ilustra bem as etapas para a implementação de filtros digitais usando uma placa de DSP acoplada ao computador IBM-PC (ver Figura 1.2).

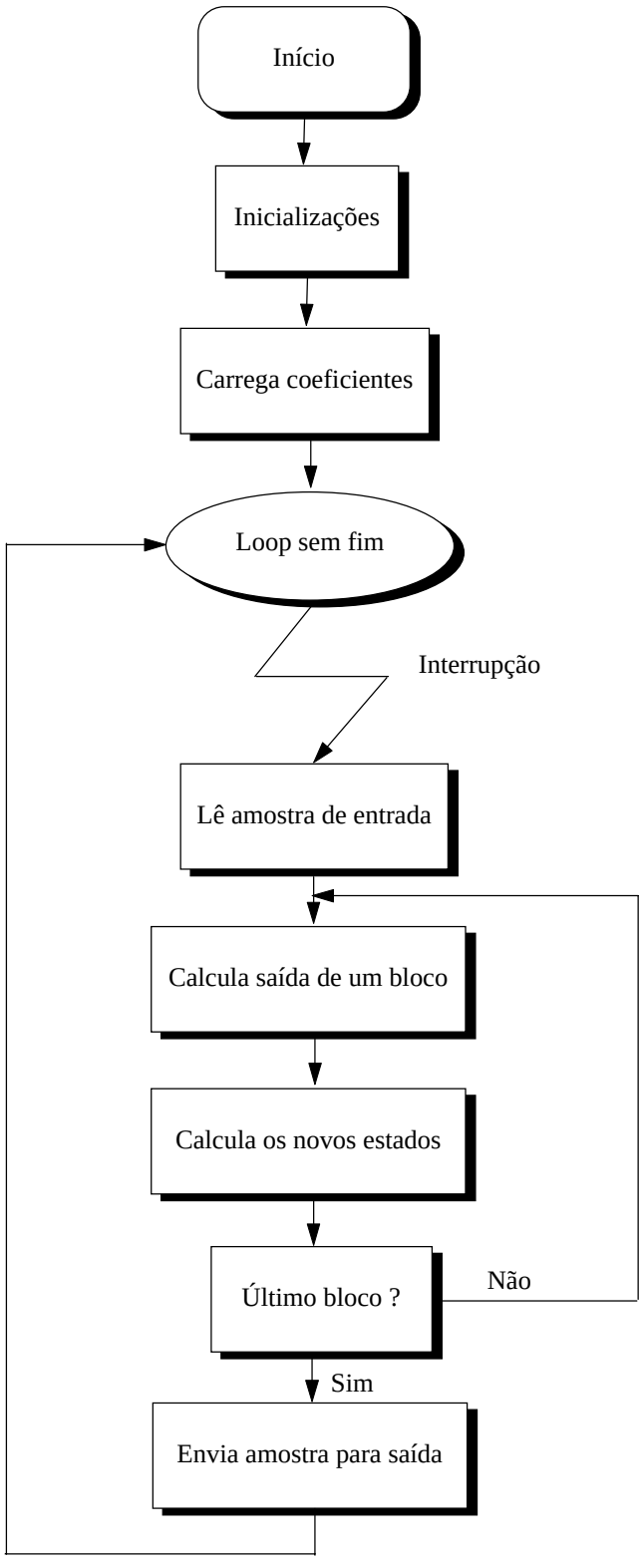


Figura 5.1. Fluxograma das etapas de implementação dos filtros no ambiente do DSP.

A próxima etapa foi desenvolver e codificar o programa fonte na linguagem assembly do TMS320C25, gerar os códigos de máquina e executar o programa objeto, o que é feito através da plataforma PIPDS. Para a estrutura da Figura 2.7, a título de exemplo, o programa fonte correspondente encontra-se no Apêndice A.

Um mapa da memória de dados do DSP, correspondente à implementação do filtro na forma paralela de seções ótimas, é apresentado na Figura 5.2. Ele mostra como os dados iniciais e os resultados intermediários e finais são armazenados.

ENDEREÇOS em hexadecimal	**NOME DA VARIÁVEL**	**DESCRIÇÃO**
410h	FREQUENCIA	Freqüência de amostragem
411h	NUMBLOCOS	Número de blocos de segunda ordem
Variáveis para o primeiro bloco		
412h	A111	Multiplicador quantizado A11 da matriz A
413h	A121	" A12 "
414h	A211	" A21 "
415h	A221	" A22 "
416h	B11	Multiplicador quantizado B1 do vetor B
417h	B21	" B2 "
418h	C11F	C1F - Parte fracionária do multiplicador C1 do vetor C
419h	C11S	C1S - Sinal do multiplicador C1 do vetor C
41Ah	C11I	C1I - Parte inteira do multiplicador C1 do vetor C
41Bh	C21F	C2F - Parte fracionária do multiplicador C2 do vetor C
41Ch	C21S	C2S - Sinal do multiplicador C2 do vetor C
41Dh	C21I	C2I - Parte inteira do multiplicador C2 do vetor C
41Eh a 429h		**Variáveis para o segundo bloco**
42Ah a 435h		**Variáveis para o terceiro bloco**
436h a 441h		**Variáveis para o quarto bloco**
442h a 44Dh		**Variáveis para o quinto bloco**
44Eh	D	Multiplicador quantizado D da rede paralela
44Fh a 458h	S1, S2	Estados de cada bloco de segunda ordem
459h	ONE	Variável auxiliar
45Ah e 45Bh	XN e YN	Entrada e saída do filtro
45Ch	ENDERECOFINAL SAIDA	Endereço final do buffer de saída

45Dh	VARTEMPORARIA	Variável temporária
45Eh	ENDERECOFINAL ENTRADA	Endereço final do buffer de entrada

Figura 5.2. Mapa da memória de dados para o exemplo abordado.

Em especial, note-se que os coeficientes c_{in} foram divididos em três valores: c_{inF}, correspondente à mantissa, c_{inS}, caracterizando o sinal, e c_{inI}, caracterizando o módulo da parte inteira de c_{in}. Isso é feito para lidar com a possibilidade de se realizar multiplicações com coeficientes maiores que um, em módulo, como discutido no Capítulo 3. Observe-se, também, que o desenvolvimento do programa fonte na linguagem "assembly" do TMS320C25 não é uma tarefa corriqueira, como pode ser visto no fluxograma da Figura 1.3. Assim, a plataforma PIPDS inclui algumas ferramentas para auxiliar o programador nessa tarefa. Uma é a rotina de "disassembly", que permite ao programador avaliar a correção do programa carregado em memória, quer pela sua execução com "breakpoints", que é uma ferramenta de projeto indispensável, quer pela visualização dos conteúdos da memória e dos registradores do DSP.

Uma outra ferramenta importante disponibilizada na plataforma é a resposta ao impulso do filtro implementado. Sua transformada de Fourier pode ser plotada, de forma que o projetista pode verificar se o filtro programado atende à especificação inicialmente estabelecida para o projeto. Para implementar tal tarefa, foi criado um buffer na memória de dados do DSP, onde h(n) é armazenado, até o limite de 2048 amostras, e então $|H(e^{j\omega T})|$ é calculado e plotado. Observe-se que para gerar h(n) o programa "assembly" é ligeiramente modificado, pois não há mais entrada externa u(n) lida em sincronismo com o tempo de amostragem. Os resultados, para a implementação do exemplo, são apresentados nas Figuras 5.3 e 5.4. A visualização de detalhes dos gráficos, como na Figura 5.4, também é possível, usando a plataforma PIPDS.

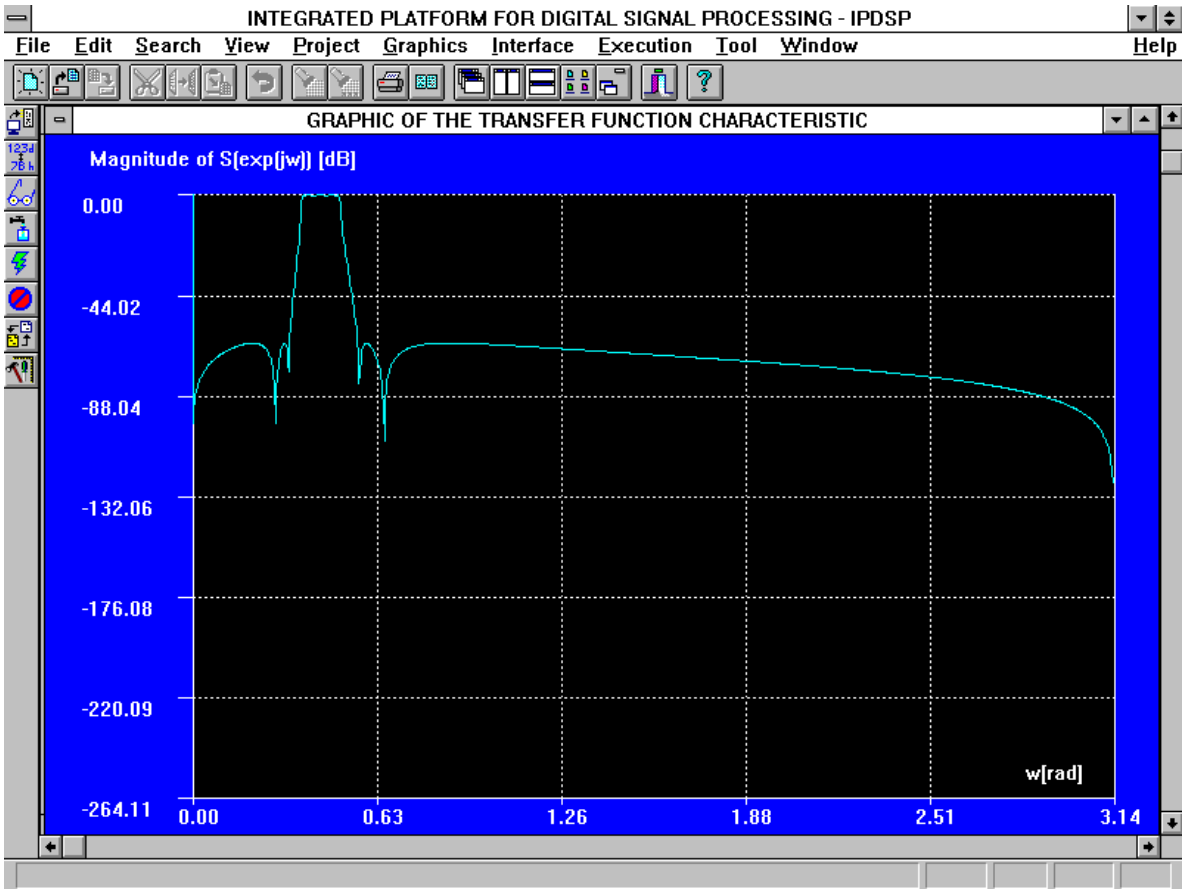


Figura 5.3.a. Resposta de magnitude do filtro passa-faixa Elíptico ideal.

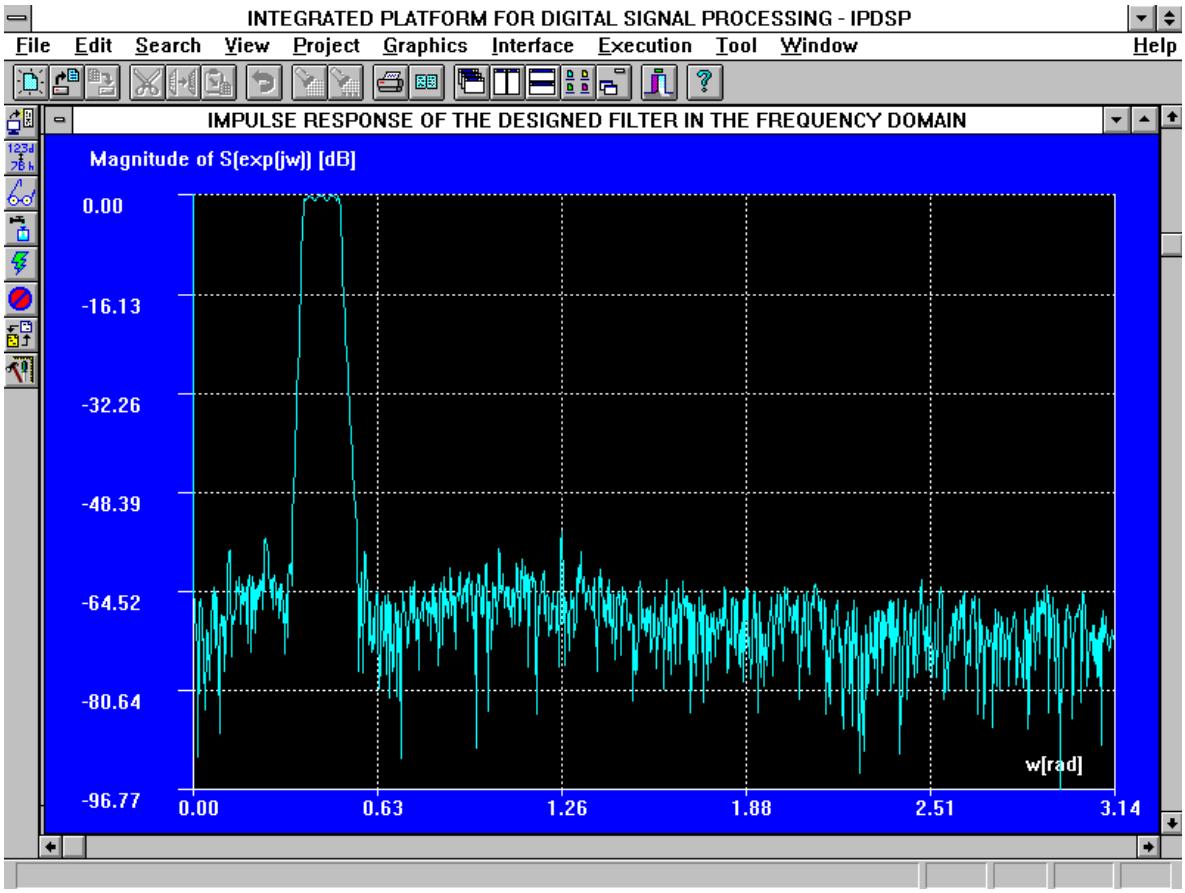


Figura 5.3.b. Magnitude da transformada de Fourier de h(n) para o filtro passa-faixa Elíptico real implementado através de blocos em paralelo.

Figura 5.3. Resposta de magnitude do filtro passa-faixa Elíptico implementado através de blocos em paralelo a) ideal b) real.

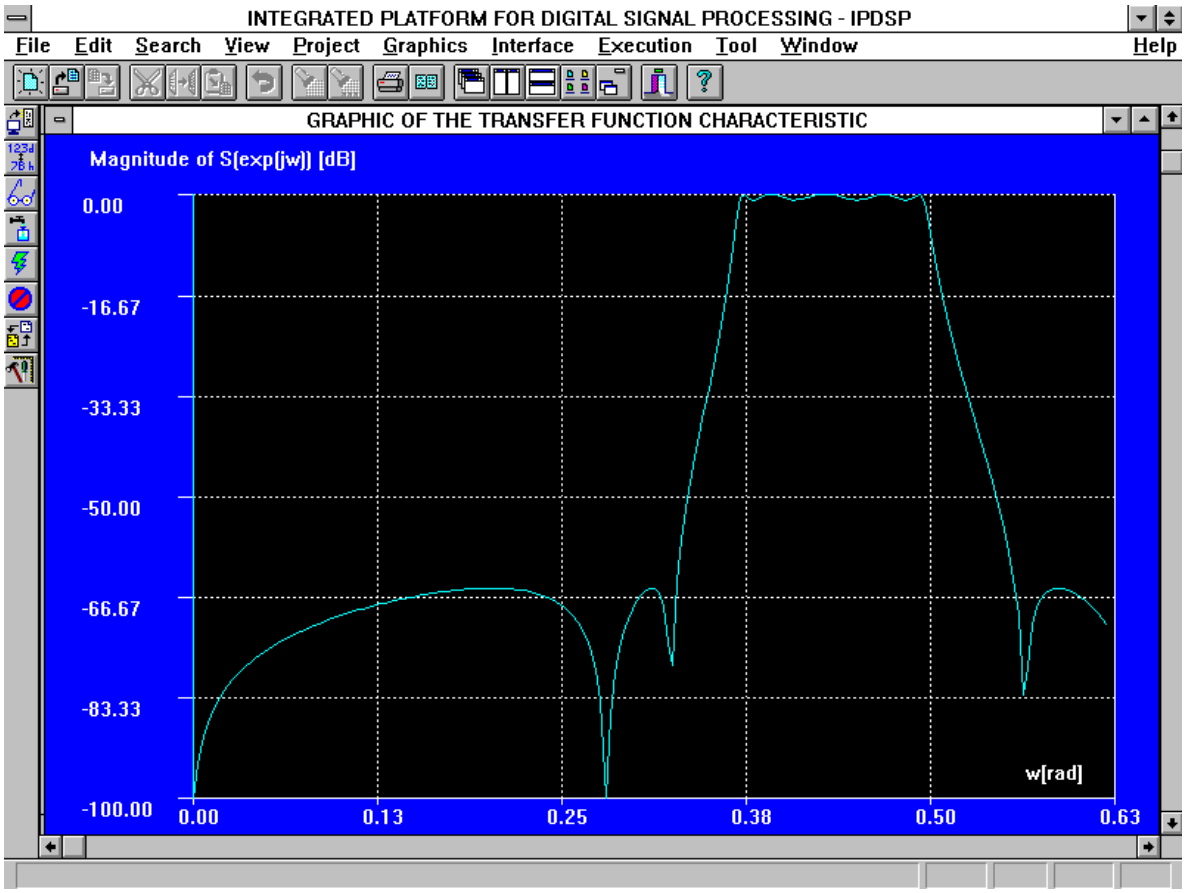


Figura 5.4.a. Detalhe da banda passante do filtro da Figura 5.3.a.

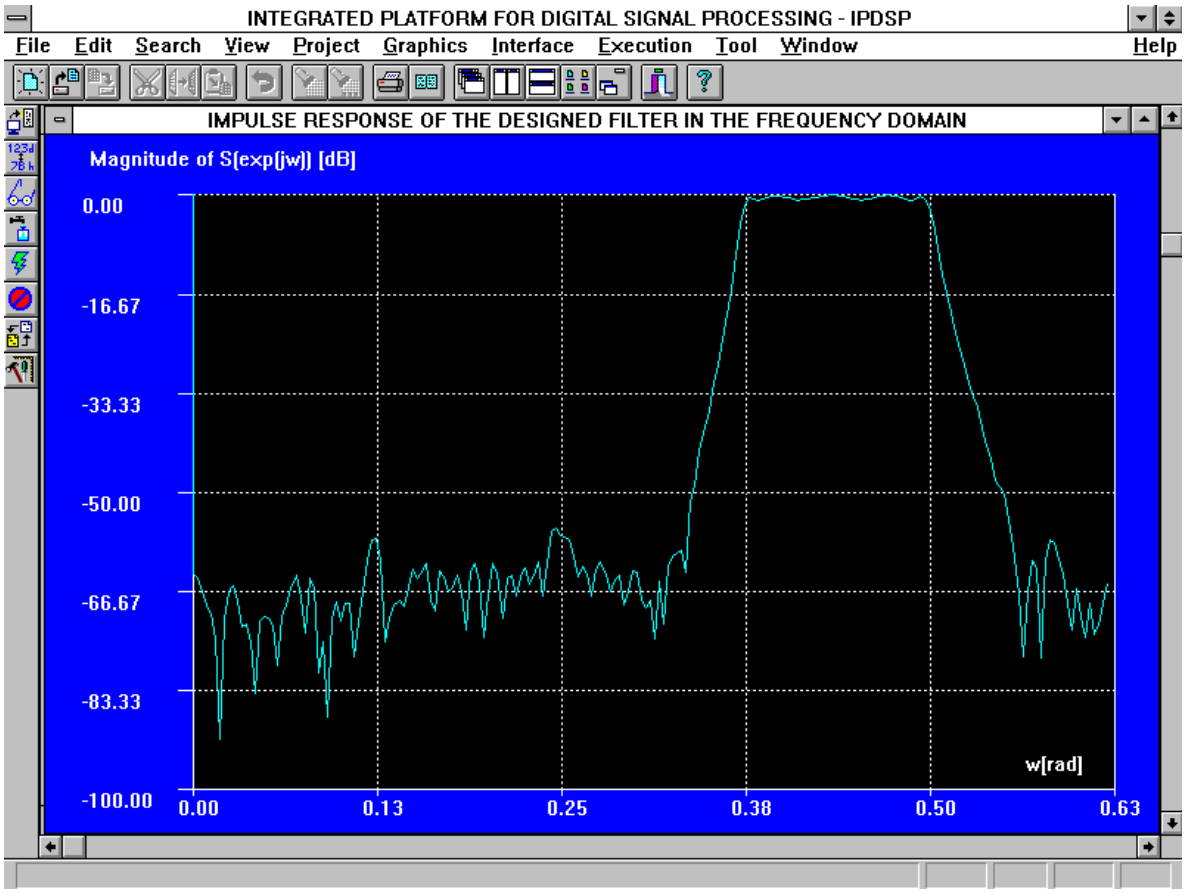


Figura 5.4.b. Detalhe da banda passante do filtro da Figura 5.3.b.

Figura 5.4. Detalhe da banda passante do filtro implementado através de blocos em paralelo a) ideal e b) real.

Conforme visto através das Figuras 5.3 e 5.4, o resultado obtido obedeceu às especificações do projeto.

Para implementar o filtro acima através de um programa “assembly” para o processador TMS320C25, foi desenvolvida uma interface que possibilita transferir os coeficientes da rede sintetizada, após sua quantização para 16 bits, para a memória local do processador, assim como programar a freqüência de amostragem do filtro. Uma outra característica importante é que cada uma das estruturas demanda um programa “assembly” específico, haja visto as diferenças estruturais entre elas. Entretanto, dada uma determinada estrutura, não há restrição nenhuma aos coeficientes do filtro a ser implementado, podendo-se implementar filtros de qualquer

espectro. Também deve ser destacado que o tratamento numérico dos sinais, na execução dos filtros projetados, faz uso de somadores de dupla precisão, truncamento em magnitude e saturação aritmética, facilmente implementáveis na linguagem assembly do TMS320C25, e que garantem a imunidade a ciclos limite dos filtros projetados /Jac79, Din86, Sar92/.

Assim, a plataforma PIPDS permite a implementação de filtros digitais de características distintas. Neste ponto, salienta-se a importância da avaliação de desempenho de cada estrutura de filtro, levando-se em conta sua complexidade computacional, medida pelo tempo necessário ao seu processamento. Há, assim, uma ferramenta adicional disponível na plataforma PIPDS para a comparação do desempenho das estruturas implementadas, a nível de tempo de processamento: o usuário define um endereço inicial e um endereço final, e o tempo gasto na execução do trecho de programa assim assinalado é medido, em ciclos de relógio, usando o temporizador interno do TMS320C25 /Tex93/.

É necessário medir o tempo de processamento das estruturas devido à arquitetura "Harvard", adotada pela maioria dos processadores digitais de sinais: como há espaços de memória e barramentos independentes para dados e programas, permitindo um alto grau de paralelismo nas operações, os ciclos de busca e de execução de uma instrução podem ocorrer simultaneamente, tornando inadequada a simples contagem de ciclos a partir do programa fonte. O tempo de processamento, é medido desde o momento em que o sinal de entrada está disponível (a partir da instrução imediatamente após à instrução `in XN, PORTAAD`) até o momento em que é enviado pela porta de saída do DSP (até a instrução imediatamente anterior à instrução `out YN, PORTADA`).

Finalmente, com o objetivo de permitir ao usuário a visão completa da operação em tempo real do filtro por ele projetado, a plataforma PIPDS incorpora duas importantes ferramentas, a saber os gráficos no tempo e em freqüência dos sinais de entrada ou saída do filtro realizado (isoladamente ou em conjunto). Para implementar tais procedimentos, foi criada uma tarefa temporizada, para o "refresh" da tela gráfica a intervalos regulares, baseada nos valores armazenados em dois "buffers", novamente de 2048 amostras, implementados em forma de fila circular (para permitir à interface gráfica controlar a posição da amostra mais recente, já que o temporizador de "refresh" da tela gráfica não está sincronizado com o tempo de amostragem do filtro) na memória do DSP, e que são lidos periodicamente pela interface gráfica.

Para o exemplo descrito neste capítulo, os gráficos das Figuras 5.5.a e b ilustram a resposta no tempo e o espectro dos sinais de entrada e saída.

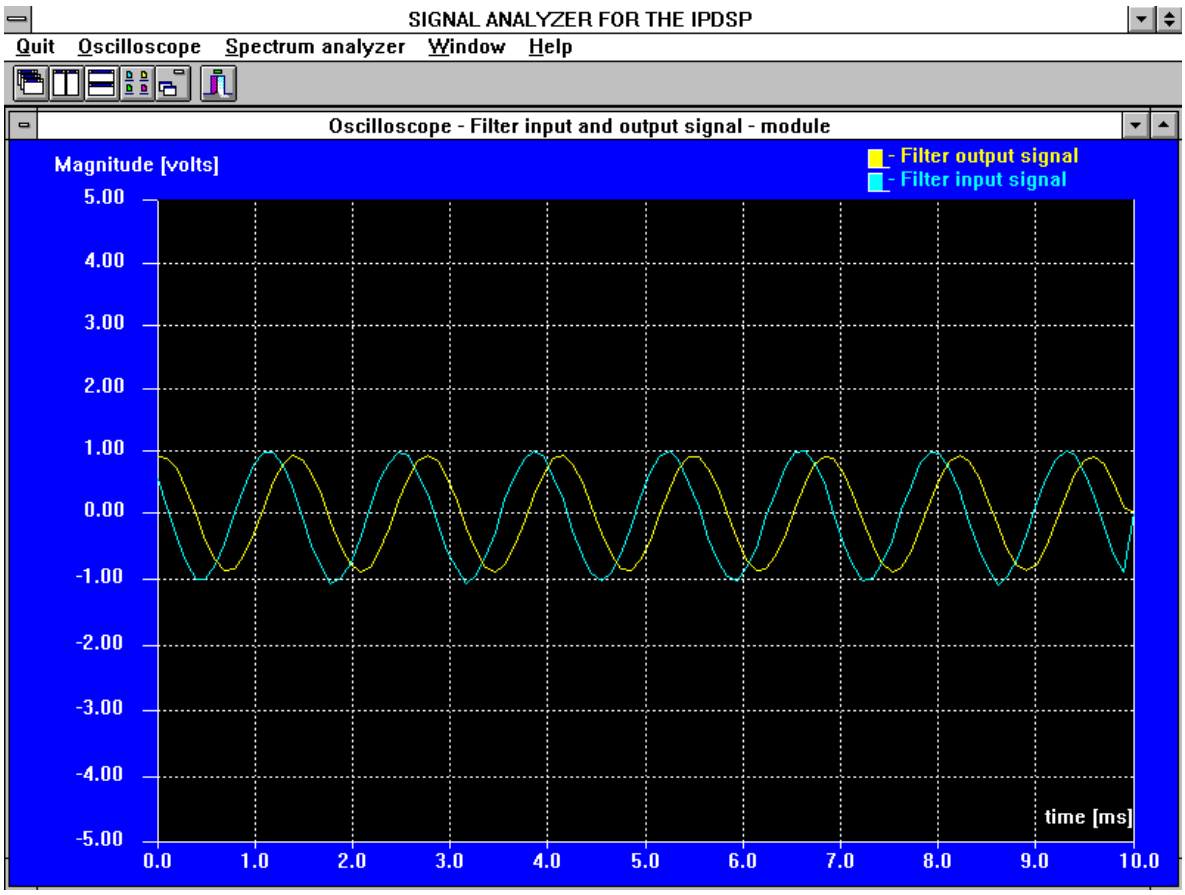


Figura 5.5.a. Gráficos de x(n) e y(n).

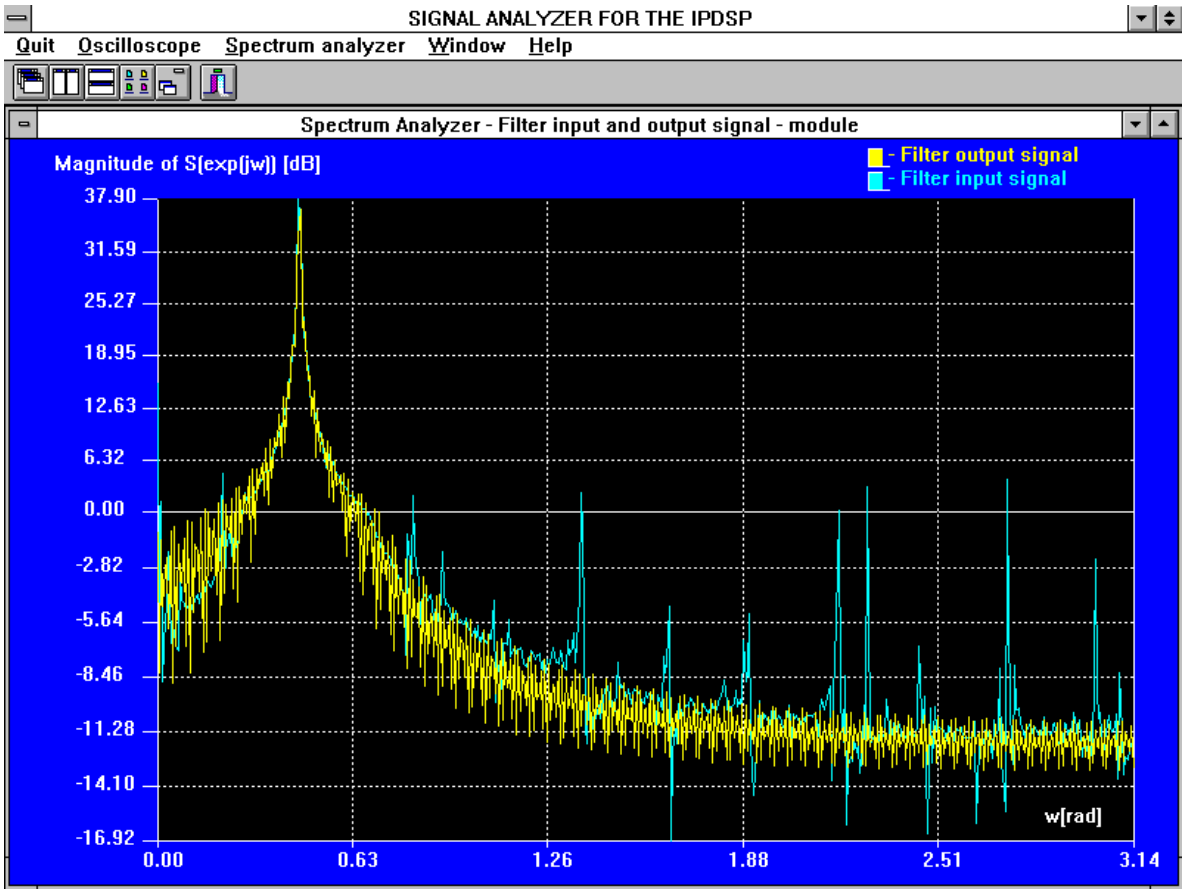


Figura 5.5.b. Espectros dos sinais de entrada e saída (módulo).

Figura 5.5. Caracterização no tempo e em freqüência do exemplo implementado.

Com o exemplo aqui abordado, foi possível ilustrar o uso da plataforma PIPDS, mostrando assim os aspectos básicos de sua concepção e suas possibilidades no apoio ao projetista de filtros digitais. Para melhor entendimento, sugere-se que este capítulo seja lido ao mesmo tempo em que o exemplo é desenvolvido, diretamente na plataforma. De qualquer forma, a plataforma PIPDS foi concebida de forma a facilitar

seu uso, como pode ser notado pela frequente utilização de caixas de diálogo, assim como pela utilização de textos de ajuda.

CAPÍTULO 6

Conclusões e Sugestões para Trabalhos Futuros

Neste trabalho foi apresentada uma Plataforma Integrada para Processamento Digital de Sinais, denominada PIPDS, concebida como um ambiente integrado para o projeto e a implementação, em DSP, de filtros digitais. O sistema incorpora, como principais características, a facilidade de análise no tempo e em freqüência dos sinais envolvidos no processo de filtragem, além da possibilidade da programação de famílias inteiras de filtros digitais, através de uma interface adequada entre o processador do computador PC hospedeiro e o DSP, responsável pela transferência dos coeficientes do filtro particular projetado, assim como pela correta programação do período de amostragem. Além disso, a plataforma PIPDS permite ao usuário realizar toda a tarefa de projeto dos filtros a serem implementados.

O amplo leque de opções oferecido por este software possibilita ao usuário, com grande flexibilidade e eficiência, uma rápida avaliação do projeto proposto. Um exemplo, ilustrativo da utilização desta plataforma foi mostrado, comprovando sua versatilidade e eficiência.

A plataforma PIPDS é um ambiente Windows de desenvolvimento, de aplicação genérica em filtragem digital. A simplicidade de sua arquitetura e a utilização de um microcomputador compatível com IBM-PC como estação de trabalho, este já bastante difundido nas empresas e nos laboratórios de pesquisa, a credenciam como um ambiente de projeto e implementação de filtros digitais, ou mesmo de outros algoritmos para diversas áreas da engenharia que desejem aplicar técnicas de processamento digital de sinais. A plataforma PIPDS pode também ser utilizada como base para pesquisas em áreas que empreguem técnicas de processamento digital de sinais, tais como ensaios de vibrações mecânicas, processamento de sinais de eco em aplicações de ultra-som, etc.

No sentido da continuidade do desenvolvimento da plataforma, sugere-se como possível evolução deste trabalho:

- a criação de uma interface mais amigável que permita ao usuário escolher a forma do filtro a ser implementado (cascata ou paralela) e os blocos básicos desejados (incluindo seu número), sendo o programa fonte correspondente ao filtro assim descrito gerado automaticamente;

- o desenvolvimento do sistema com DSP’s de quarta ou quinta geração, baseando-se nos processadores de ponto flutuante, utilizando-se os conceitos desenvolvidos nesse trabalho;

- o desenvolvimento de sistemas de interligação de vários processadores, visando a possibilidade de implementar algoritmos complexos dividindo as tarefas entre os processadores. Neste contexto, os processadores teriam uma concepção simples, como a que foi usada na plataforma PIPDS e, quando em funcionamento simultâneo, permitiriam a implementação de sistemas como modens de alta velocidade, bancos de filtros, etc.;

- a migração para plataformas não-Intel tais como DEC Alpha ou Motorola PowerPC (ambas máquinas com processadores de 32 bits) rodando o Windows NT, ou mesmo explorar o endereçamento e os registradores de 32 bits dos processadores atuais baseados no chip Intel (80386 de 32 bits e superior). Inclusive, uma versão da plataforma PIPDS para estes últimos processadores, usando o ambiente Windows 95 (32 bits), já está em fase final de implementação. Deve ser destacado que o código Win16 utilizado nesta versão da plataforma pode ser facilmente convertido para a API Win32. É importante observar que, muito embora a interface gráfica baseada no Windows NT possa ser utilizada com certa facilidade, a plataforma PIPDS também deve ser atualizada em função da placa de DSP usada, até para considerar as particularidades de barramento, quando forem usadas plataformas não-Intel.

Referências Bibliográficas

Ant93 A. Antoniou, “Digital filters: Analysis, Design and Applications”, Second Edition, McGraw & Hill Book company, USA, 1993.

Asp84 Digital Filter Design Package, User’s Handbook, Atlanta Signal Processors Incorporated, 1984.

Bau93 P. H. Bauer e J. Wang, “Limit Cycle Bounds for Floating Point Implementations of Second-Order Recursive Digital Filters”, IEEE Transactions on Circuits and Systems - II: Analog and Digital Signal Processing, vol. 40, pp. 293-301, agosto de 1993.

Bau95 P. H. Bauer, “Absolute Response Error Bounds for Floating-Point Digital Filters in State Space Representation”, IEEE Transactions on Circuits and Systems - II: Analog and Digital Signal Processing, vol. 41, pp. 610-613, Setembro de 1995.

Bom85 Bruce W. Bomar, “New Second-Order State-Space Structures for Realizing Low Roundoff Noise Digital Filters”, IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 33, pp. 106-110, fevereiro de 1985.

Bom89 Bruce W. Bomar, “On the Design of Second-Order State-Space Digital Filter Sections”, IEEE Transactions on Circuits and Systems, vol. 36, pp. 542-552, abril de 1989.

Bom94 B. W. Bomar, “Low-Roundoff-Noise Limit-Cycle-Free Implementation of Recursive Transfer Functions on a Fixed-Point Digital Signal Processor”, IEEE Transactions on Industrial Electronics, vol. 41, pp. 70-78, fevereiro de 1994.

But75 H. J. Butterweck, “Suppression of Parasitic Oscillations in Second-Order Digital Filters by Means of a Controlled-Rounding Arithmetic”, Arch. Elek. Ubertragung, vol. 29, pp.371-374, 1975.

Cha90 R. Chassaing and D. W. Horning, “Digital Signal Processing with the TMS320C25”, John Wiley & Sons, USA, 1990.

Cro75 R. E. Crochiere e A. V. Oppenheim, “Analysis of Linear Digital Networks”, Proc. IEEE, vol. 63, pp. 581-595, abril de 1975.

Dal90 Dalanco Spry, “Model 250 Data Acquisition and Signal Processing Board for the IBM PC AT and ISA Bus Compatibles”, Rochester/NY, USA, 1990.

Din86 P. S. R. Diniz e A. Antoniou, “More Economical State-Space Digital-Filter Structures Which Are Free of Constant-Input Limit Cycles”, IEEE Transactions on Acoustics, Speech and Signal Processing, vol ASSP-34, pp. 807-815, agosto de 1986.

Ess86 D. Essig, C. Erskine, E. Cauldel e S. Magar, “A Second Generation Digital Signal Processor”, IEEE Trans. on Circuits an Systems, CAS-33, pp. 196-200, fevereiro de 1986.

Fai85 R. Fairley, “Software Engineering Concepts”, McGraw-Hill Book Company, USA, 1985.

Jac70 L. B. Jackson, “Roundoff-Noise Analyses for Fixed-Point Digital Filters Realized in Cascade or Parallel Form”, IEEE Trans. Audio Electroacoust., vol. AU-15, pp. 107-122, junho de 1970.

Jac79 L. B. Jackson, A. G. Lindgren, e Y. Kim, “Optimal Synthesis of Second-Order State-Space Structures for Digital Filters”, IEEE Trans. Circuits Syst. vol. CAS-26, pp. 149-153, março de 1979.

Kan73 T. Kaneko, “Limit-Cycle Oscillations in Floating Point Digital Filters”, IEEE Transactions on Audio and Electroacoustics, vol. AU-21, number 2, pp. 100-106, abril de 1973.

Kie77 R. B. Kieburtz, V. B. Lawrence, and K. V. Mina, “Control of Limit Cycles in Recursive Digital Filters by Randomized Quantization”, IEEE Trans. Circuits Syst. vol. CAS-24, pp. 291-299, junho de 1977.

Kim94 S. Kim e W. Sung, “A Floating-Point to Fixed-Point Assembly Program Translator for the TMS320C25”, IEEE Transactions on Circuits and Systems - II: Analog and Digital Signal Processing, vol. 41, pp. 730-739, novembro de 1994.

Laa87 T. Laakso, “Comparison of Signal Processor Implementations of Recursive Digital Filter Structures”, Tese de Mestrado, Helsinki University of Technology, 1987.

Laa94a T. Laakso, B. Zeng, L. Hartimo e Y. Neuvo, “Elimination of Limit Cycles in Floating-Point Implementations of Recursive Digital Filters”, IEEE Transactions on Circuits and Systems - II: Analog and Digital Signal Processing, vol. 41, pp. 308-313, abril de 1994.

Laa94b T. Laakso, “Bounds for Floating-Point Roundoff Noise”, IEEE Transactions on Circuits and Systems - II: Analog and Digital Signal Processing, vol. 41, pp 424-426, junho de 1994.

Mil78 W. L. Mills, C. T. Mullis, e R. A. Roberts, “Digital Filters Without Overflow Oscillations”, IEEE Trans. Acoust. Speech Sign Process., vol. ASSP-26, pp. 334-338, agosto de 1978.

Mor86 L. R. Morris (editor), “Digital Signal Processing Microprocessors: Forward to the Past?”, IEEE Micro, vol. 6, no. 6, dezembro de 1986.

Mul76a C. T. Mullis e R. A. Roberts, “Synthesis of Minimum Roundoff Noise Fixed Point Digital Filters”, IEEE Transactions on Circuits and Systems, vol. CAS-23, pp 551-562, setembro de 1976.

Mul76b C. T. Mullis e R. A. Roberts, “Roundoff Noise in Digital Filters: Frequency Transformations and Invariants”, IEEE Transactions on Acoustics, Speech and Signal Processing, vol. ASSP-24, pp. 538-550, dezembro de 1976.

Opp89 A. V. Oppenheim e R. W. Schafer, “Discrete-Time Signal Processing”, Prentice-Hall International, Inc., Englewood Cliffs, NJ, USA, 1989.

Pap84 A. Papoulis, “Probability, Random Variables, and Stochastic Processes”, McGraw-Hill Book Company, USA, 1984.

Per88 J. A. B. Pereira, J. C. E. Cabezas, P. S. R. Diniz e R. G. Lins, “PACPDS: Um Pacote de Apoio ao Projeto de Sistemas de Processamento Digital de Sinais”, Anais do VII Congresso Brasileiro de Automática, São José dos Campos/SP, pp. 267-272, 1988.

Pet93 C. Petzold, “Programando para Windows 3.1”, Makron Books do Brasil Editora, São Paulo/SP, 1993.

Pre82 R. S. Pressman, “Software Engineering: A Practitioner’s Approach”, McGraw-Hill Book Company, USA, 1982.

Rab75 L. R. Rabiner and B. Gold, “Theory and Application of Digital Signal Processing”, Prentice-Hall, New Jersey 1975.

Ren82 M. Renfors, “On the Design of Efficient Digital Filters Using Graph Theoretic Equivalence Transformations”, Tese de Doutorado, Dept. Elec. Eng., Tampere Univ. of Technology, Tampere, Finlândia, 1982.

Sar90 M. Sarcinelli Filho, “Síntese de Filtros Digitais Recursivos Sem Ciclos Limite”, Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro/RJ, 1990.

Sar92 M. Sarcinelli Filho, C. P. da Cruz, A. C. S. Simmer e P. S. R. Diniz, “Estrutura Digital de Segunda Ordem Quase Ótima Livre de Ciclos Limite”, Anais do IX Congresso Brasileiro de Automática, Vitória/ES, pp. 40-44, setembro de 1992.

Sar94 M. Sarcinelli Filho, A. C. Machado e L. G. Ferreira, “Implementação de Filtros Digitais Usando Processador Dedicado”, Anais do XVII Congresso Nacional de Matemática Aplicada e Computacional, Vitória/ES, pp. 338-342, 1994.

Sim94 A. C. S. Simmer, “Análise Comparativa de Algumas Estruturas Digitais de Segunda Ordem Imunes a Ciclos Limite”, Dissertação de Mestrado, UFES, Vitória/ES, 1994.

Sin85 V. Singh, “A New Realizability Condition for Limit Cycle-Free State-Space Digital Filters Employing Saturation Arithmetic”, IEEE Transactions on Circuits and Systems, vol. CAS-32, pp. 1070-1071, outubro de 1985.

Sta84 W. D. Stanley and R. Dougherty, “Digital Signal Processing”, Reston Publishing Company Inc., Virginia, 1984.

Swa94 T. Swam, “Advanced Programming Using Borland C++ 4.02 for Windows”, Berkeley Brazil Publishing Company, São Paulo/SP, 1994.

Tex83 Texas Instruments , “TMS 32010 User’s Guide”, USA, 1983.

Tex86 Texas Instruments, “Digital Signal Processing Applications With TMS 320 Family: Theory, Algorithms and Implementations”, USA, 1986.

Tex93 Texas Instruments, “TMS320C2X User’s Guide”, USA, 1993.

Vai87 P. P. Vaydianathan e V. Liu, “An Improved Sufficient Condition for Absence of Limit Cycles in Digital Filters”, IEEE Transactions on Circuits and Systems, vol. CAS-34, pp. 319-322, março de 1987.

Yao95 P. Yao, “Borland C++ 4.0 Programação for Windows”, Makron Books do Brasil Editora, São Paulo/SP, 1995.

Apêndice A

Código "assembly" para Implementação em DSP da Estrutura na Forma Paralela de Blocos Quadráticos de Mínimo Ruído

```
;************************************************
;* REDE OTIMA - FORMA PARALELA - MAXIMO 5 BLOCOS *
;************************************************

IniciodoBufferentrada   equ         >1001 ; Endereco inicial do buffer
FimdoBufferentrada      equ         >1802 ; Endereco final do buffer + 1
IniciodoBuffersaida     equ         >2001 ; Endereco inicial do buffer
FimdoBuffersaida        equ         >2802 ; Endereco final do buffer + 1

;
; PARAMETROS DA PLACA
;

PORTAAD                 equ         2
PORTADA                 equ         3
DAC0                    equ         0
DAC1                    equ         >8
TIMER                   equ         0
LATCH                   equ         1
MASCARA                 equ         2
ACKPT                   equ         0
CANAL0                  equ         0
CANAL5                  equ         5
TEMP                    equ         77
ZERO                    equ         78
AUX                     equ         80

;
; Inicio do segmento de dados
;

dseg

FREQUENCIA              bss         1
NUMBLOCOS               bss         1
A111                    bss         1
A121                    bss         1
A211                    bss         1
A221                    bss         1
B11                     bss         1
B21                     bss         1
C11f                    bss         1
C11s                    bss         1
C11i                    bss         1
C21f                    bss         1
C21s                    bss         1
C21i                    bss         1
A112                    bss         1
A122                    bss         1
A212                    bss         1
A222                    bss         1
B12                     bss         1
B22                     bss         1
C12f                    bss         1
C12s                    bss         1
C12i                    bss         1
C22f                    bss         1
C22s                    bss         1
C22i                    bss         1
A113                    bss         1
A123                    bss         1
A213                    bss         1
```

```
B13                     bss         1
C13f                    bss         1
C13i                    bss         1
C23f                    bss         1
C23s                    bss         1
C23i                    bss         1
A114                    bss         1
A124                    bss         1
A214                    bss         1
A224                    bss         1
B14                     bss         1
B24                     bss         1
C14f                    bss         1
C14s                    bss         1
C14i                    bss         1
C24f                    bss         1
C24s                    bss         1
C24i                    bss         1
A115                    bss         1
A125                    bss         1
A215                    bss         1
A225                    bss         1
B15                     bss         1
B25                     bss         1
C15f                    bss         1
C15s                    bss         1
C15i                    bss         1
C25f                    bss         1
C25s                    bss         1
C25i                    bss         1
D                       bss         1

S11                     bss         1
S21                     bss         1
S12                     bss         1
S22                     bss         1
S13                     bss         1
S23                     bss         1
S14                     bss         1
S24                     bss         1
S15                     bss         1
S25                     bss         1

ONE                     bss         1

XN                      bss         1
YN                      bss         1

EnderecoFinalsaida      bss         1
VarTemporaria           bss         1
EnderecoFinalentrada    bss         1

dend


                aorg        0

                b           INICIO
                aorg        4

                b           PRINCIPAL
```

```
;
; ROTINA EXECUTA TRUNCAMENTO EM MAGNITUDE
;

TRUNC             bgez        MAGN
                  add         ONE,14
                  MAGN        ret

;
; INICIALIZACAO DA PLACA
;

INICIO            dint
                  ldpk        0
                  lack        MASCARA
                  sacl        4
                  ldpk        7

                  lalk        FimdoBufferentrada
                  sacl        EnderecoFinalentrada

                  lalk        FimdoBuffersaida
                  sacl        EnderecoFinalsaida

                  lrlk        2,>410
                  lrlk        3,IniciodoBuffersaida
                  lrlk        4,>1000
                  lrlk        5,IniciodoBufferentrada
                  lrlk        6,>2000

                  spm         0

                  larp        2
                  lac         *+
                  sacl        FREQUENCIA
                  lac         *+
                  sacl        NUMBLOCOS
                  lac         *+
                  sacl        A111
                  lac         *+
                  sacl        A121
                  lac         *+
                  sacl        A211
                  lac         *+
                  sacl        A221
                  lac         *+
                  sacl        B11
                  lac         *+
                  sacl        B21
                  lac         *+
                  sacl        C11f
                  lac         *+
                  sacl        C11s
                  lac         *+
                  sacl        C11i
                  lac         *+
                  sacl        C21f
                  lac         *+
                  sacl        C21s
                  lac         *+
                  sacl        C21i
                  lac         *+
                  sacl        A112
                  lac         *+
                  sacl        A122
```

```
                lac     *+
                sacl    A212
                lac     *+
                sacl    A222
                lac     *+
                sacl    B12
                lac     *+
                sacl    B22
                lac     *+
                sacl    C12f
                lac     *+
                sacl    C12s
                lac     *+
                sacl    C12i
                lac     *+
                sacl    C22f
                lac     *+
                sacl    C22s
                lac     *+
                sacl    C22i
                lac     *+
                sacl    A113
                lac     *+
                sacl    A123
                lac     *+
                sacl    A213
                lac     *+
                sacl    A223
                lac     *+
                sacl    B13
                lac     *+
                sacl    B23
                lac     *+
                sacl    C13f
                lac     *+
                sacl    C13s
                lac     *+
                sacl    C13i
                lac     *+
                sacl    C23f
                lac     *+
                sacl    C23s
                lac     *+
                sacl    C23i
                lac     *+
                sacl    A114
                lac     *+
                sacl    A124
                lac     *+
                sacl    A214
                lac     *+
                sacl    A224
                lac     *+
                sacl    B14
                lac     *+
                sacl    B24
                lac     *+
                sacl    C14f
                lac     *+
                sacl    C14s
                lac     *+
                sacl    C14i
                lac     *+
                sacl    C24f
                lac     *+
```

```
                    sacl        C24s
                    lac         *+
                    sacl        C24i
                    lac         *+
                    sacl        A115
                    lac         *+
                    sacl        A125
                    lac         *+
                    sacl        A215
                    lac         *+
                    sacl        A225
                    lac         *+
                    sacl        B15
                    lac         *+
                    sacl        B25
                    lac         *+
                    sacl        C15f
                    lac         *+
                    sacl        C15s
                    lac         *+
                    sacl        C15i
                    lac         *+
                    sacl        C25f
                    lac         *+
                    sacl        C25s
                    lac         *+
                    sacl        C25i
                    lac         *+
                    sacl        D       ; MULTIPLICADOR ENTRADA/SAIDA
                    lack        0
                    sacl        S11
                    sacl        S21
                    sacl        S12
                    sacl        S22
                    sacl        S13
                    sacl        S23
                    sacl        S14
                    sacl        S24
                    sacl        S15
                    sacl        S25

                    lack        1
                    sacl        ONE
                    lac         FREQUENCIA
                    sacl        TEMP
                    out         TEMP,TIMER
                    lack        CANAL5
                    ork         DAC0
                    sacl        TEMP
                    out         TEMP,LATCH
                    rsxm
                    sovm
                    spm         1
                    larp        1
                    eint

;
; INTERRUPCAO - ESPERA
;

ESPERA              b           ESPERA
```

```
;
; CONVERSAO DE XN PARA O FORMATO Q15
;

PRINCIPAL         in          XN,PORTAAD
                  lac         XN,4
                  sub         ONE,15
                  sacl        XN

;
;DETERMINA YN
;

DeterminaYN:      zac
                  lt          S11
                  mpy         C11f

                  lar         0,C11s
                  lark        1,0
                  cmpr        0
                  bbnz        J1
                  lark        1,1
                  cmpr        0
                  bbnz        MULSOMA1
                  rpt         C11i
                  subh        S11
                  b           J1
MULSOMA1          rpt         C11i
                  addh        S11

J1                lta         S21
                  mpy         C21f

                  lar         0,C21s
                  lark        1,0
                  cmpr        0
                  bbnz        J2
                  lark        1,1
                  cmpr        0
                  bbnz        MULSOMA2

                  rpt         C21i
                  subh        S21
                  b           J2
MULSOMA2          rpt         C21i
                  addh        S21

J2                lta         S12
                  mpy         C12f

                  lar         0,C12s
                  lark        1,0
                  cmpr        0
                  bbnz        J3
                  lark        1,1
                  cmpr        0
                  bbnz        MULSOMA3

                  rpt         C12i
                  subh        S12
                  b           J3

MULSOMA3          rpt         C12i
                  addh        S12
```

```
J3              lta         S22
                mpy         C22f

                lar         0,C22s
                lark        1,0
                cmpr        0
                bbnz        J4
                lark        1,1
                cmpr        0
                bbnz        MULSOMA4

                rpt         C22i
                subh        S22
                b           J4

MULSOMA4        rpt         C22i
                addh        S22

J4              lta         S13
                mpy         C13f

                lar         0,C13s
                lark        1,0
                cmpr        0
                bbnz        J5
                lark        1,1
                cmpr        0
                bbnz        MULSOMA5

                rpt         C13i
                subh        S13
                b           J5

MULSOMA5        rpt         C13i
                addh        S13

J5              lta         S23
                mpy         C23f

                lar         0,C23s
                lark        1,0
                cmpr        0
                bbnz        J6
                lark        1,1
                cmpr        0
                bbnz        MULSOMA6

                rpt         C23i
                subh        S23
                b           J6

MULSOMA6        rpt         C23i
                addh        S23

J6              lta         S14
                mpy         C14f

                lar         0,C14s
                lark        1,0
                cmpr        0
                bbnz        J7
                lark        1,1
                cmpr        0
                bbnz        MULSOMA7
```

```
                  rpt         C14i
                  subh        S14
                  b           J7

MULSOMA7          rpt         C14i
                  addh        S14

J7                lta         S24
                  mpy         C24f

                  lar         0,C24s
                  lark        1,0
                  cmpr        0
                  bbnz        J8
                  lark        1,1
                  cmpr        0
                  bbnz        MULSOMA8

                  rpt         C24i
                  subh        S24
                  b           J8

MULSOMA8          rpt         C24i
                  addh        S24

J8                lta         S15
                  mpy         C15f

                  lar         0,C15s
                  lark        1,0
                  cmpr        0
                  bbnz        J9
                  lark        1,1
                  cmpr        0
                  bbnz        MULSOMA9

                  rpt         C15i
                  subh        S15
                  b           J9

MULSOMA9          rpt         C15i
                  addh        S15

J9                lta         S25
                  mpy         C25f

                  lar         0,C25s
                  lark        1,0
                  cmpr        0
                  bbnz        J10
                  lark        1,1
                  cmpr        0
                  bbnz        MULSOMA10

                  rpt         C25i
                  subh        S25
                  b           J10

MULSOMA10         rpt         C25i
                  addh        S25

J10               lta         XN
                  mpy         D

                  apac
```

```
                call        TRUNC
                sach        YN

;
; BLOCO NUMERO 1
;

                zac
                lt          XN
                mpy         B11
                lta         S21
                mpy         A121
                lta         S11
                mpy         A111

                apac
                call        TRUNC
                sach        S11

                zac
                mpy         A211
                lta         XN
                mpy         B21
                lta         S21
                mpy         A221

                apac
                call        TRUNC
                sach        S21

;
; BLOCO NUMERO 2
;

                zac
                lt          XN
                mpy         B12
                lta         S22
                mpy         A122
                lta         S12
                mpy         A112

                apac
                call        TRUNC
                sach        S12

                zac
                mpy         A212
                lta         XN
                mpy         B22
                lta         S22
                mpy         A222

                apac
                call        TRUNC
                sach        S22

;
; BLOCO NUMERO 3
;

                zac
                lt          XN
                mpy         B13
                lta         S23
```

```
                  mpy         A123
                  lta         S13
                  mpy         A113

                  apac
                  call        TRUNC
                  sach        S13

                  zac
                  mpy         A213
                  lta         XN
                  mpy         B23
                  lta         S23
                  mpy         A223

                  apac
                  call        TRUNC
                  sach        S23

;
; BLOCO NUMERO 4
;

                  zac
                  lt          XN
                  mpy         B14
                  lta         S24
                  mpy         A124
                  lta         S14
                  mpy         A114

                  apac
                  call        TRUNC
                  sach        S14

                  zac
                  mpy         A214
                  lta         XN
                  mpy         B24
                  lta         S24
                  mpy         A224
                  apac
                  call        TRUNC
                  sach        S24

;
; BLOCO NUMERO 5
;

                  zac
                  lt          XN
                  mpy         B15
                  lta         S25
                  mpy         A125
                  lta         S15
                  mpy         A115

                  apac
                  call        TRUNC
                  sach        S15

                  zac
                  mpy         A215
                  lta         XN
                  mpy         B25
```

```
                lta         S25
                mpy         A225

                apac
                call        TRUNC
                sach        S25

;
; Carrega buffers de entrada e saida
;

                zac
                lac         XN
                larp        5
                sacl        *+
                lac         EnderecoFinalentrada
                sar         5,VarTemporaria
                sub         VarTemporaria
                bnz         BufferdeYN
                lrlk        5,IniciodoBufferentrada

BufferdeYN      zac
                lac         VarTemporaria
                larp        4
                sacl        *

                zac
                lac         YN
                larp        3
                sacl        *+

                lac         EnderecoFinalsaida
                sar         3,VarTemporaria
                sub         VarTemporaria
                bnz         Continua
                lrlk        3,IniciodoBuffersaida

Continua:       zac
                lac         VarTemporaria
                larp        6
                sacl        *
                larp        0

;
; ENVIA SAIDA PELA PORTA 3
;

                zac
                lac         YN
                add         ONE,15
                sacl        YN
                lac         YN,12
                sach        YN
                out         YN,PORTADA      ;ENVIO PELA PORTA 3
                eint

                b           ESPERA

end
```